Traffic Structure-Aware Network Telemetry Systems: Foundations, Designs, and Applications

by

Christopher H. Misa

A dissertation accepted and approved in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in Computer Science

Dissertation Committee:

Ramakrishnan Durairajan, Co-Chair

Reza Rejaie, Co-Chair

Zena Ariola, Core Member

Jun Li, Core Member

Walter Willinger, Core Member

David Levin, Institutional Representative

University of Oregon

Spring 2024

DISSERTATION ABSTRACT

Christopher H. Misa

Doctor of Philosophy in Computer Science

Title: Traffic Structure-Aware Network Telemetry Systems: Foundations, Designs, and Applications

Real-time traffic monitoring is a mission-critical capability for engineers and administrators tasked with managing modern computer networks. To cope with the challenges of extremely large traffic volumes, the emergence of programmable switch hardware promises the possibility of traffic monitoring systems with high packet processing efficiency, low energy and capital costs, and the ability to produce detailed results for a wide range of tasks. However, the high efficiency of programmable switch hardware necessitates a constrained programming model with access to only a small amount of high-speed memory, a limited number of primitive operations per packet, and tens of seconds of network downtime each time the program is changed. Despite significant research effort on developing efficient traffic monitoring systems within these constraints, current approaches are critically limited in light of real-world traffic structure and task requirements.

To address these limitations and to pave the way for principled approaches in future research, we leverage the observation that real-world network traffic is not generated uniformly at random, but exhibits complex statistical structure resulting from human and machine communications. By developing characterizations of this structure, we propose a novel refocusing of state-of-the-art towards investigation of structure-aware telemetry systems to improve the efficiency and practicality of real-world traffic monitoring tasks. In particular, we develop novel contributions in

characterizing traffic structure, designing algorithms for traffic monitoring capabilities on programmable switch hardware, and leveraging these capabilities for practical real-world traffic monitoring tasks.

This dissertation includes previously published co-authored material as well as previously unpublished co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: Christopher H. Misa

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene, OR, USA
DePaul University, Chicago, IL, USA

DEGREES AWARDED:

Doctor of Philosophy, Computer Science, 2024, University of Oregon
Bachelor of Music, Composition, 2014, DePaul University

AREAS OF SPECIAL INTEREST:

Network traffic monitoring
Linux kernel network stack
Statistical analysis

PROFESSIONAL EXPERIENCE:

Graduate Research Assistant, Oregon Networking Research Group, University of Oregon, 2019-2024
Research Intern, Futurewei Technologies Inc., 2022
Undergraduate Research Assistant, Oregon Networking Research Group, University of Oregon, 2018

GRANTS, AWARDS AND HONORS:

Phillip Seeley Graduate Fellowship, University of Oregon, 2022
Ripple Fellowship, University of Oregon, 2020, 2021, 2022
Gurdeep Pall Graduate Student Fellowship, University of Oregon, 2020
Undergraduate Research Fellowship, Vice President for Research and Innovation, University of Oregon, 2018

Clarence and Lucille Dunbar Scholarship, University of Oregon, 2018
Erwin and Gertrude Juilfs Scholarship in Computer and Information Science, University of Oregon, 2018

PUBLICATIONS:

Misa, C., Durairajan, R., Gupta, A., Rejaie, R., and Willinger, W. Leveraging Prefix Structure to Detect Volumetric DDoS Attack Signatures with Programmable Switches. In *Proceedings of the IEEE Symposium on Security and Privacy* (2024), pp. 205-205.

Misa, C., Durairajan, R., Rejaie, R., and Willinger, W. DynATOS+: A Network Telemetry System for Dynamic Traffic and Query Workloads. *IEEE/ACM Transactions on Networking* (2024), pp. 1-16.

Misa, C. Designing Traffic Monitoring Systems for Self-Driving Networks (Workshop Presentation). *ACM SIGMETRICS Performance Evaluation Review 51, 2* (2023), pp. 85-87.

Misa, C., O'Connor, W., Durairajan, R., Rejaie, R., and Willinger, W. Dynamic Scheduling of Approximate Telemetry Queries. In *Proceedings of the USENIX Symposium on Network Systems Design and Implementation* (2022), pp. 701-717.

Misa, C., Durairajan, R., Rejaie, R., and Willinger, W. Revisiting Network Telemetry in COIN: A Case for Runtime Programmability. *IEEE Network, Special Issue on In-Network Computing: Emerging Trends for the Edge-Cloud Continuum* (2021), pp. 14-20.

Misa, C., Guse, D., Hohlfeld, O., Durairajan, R., Sperotto, A., Dainotti, A., Rejaie, R. Lessons Learned Organizing the PAM 2020 Virtual Conference. *ACM SIGCOMM Computer Communication Review 50, 3* (2020), pp. 46-54.

Misa, C., Kannan, S., Durairajan, R. Can We Containerize Internet Measurements? In *Proceedings of the ACM/IRTF/ISOC Applied Networking Research Workshop* (2019), pp. 52-58.

ACKNOWLEDGEMENTS

The work presented in this dissertation is not the product of a single mind working in isolation, but the collective result of many different fruitful collaborations and relationships throughout my PhD career. I will do my best to acknowledge individuals and organizations that seem most relevant here, but also extend my sincere gratitude to any whom I may have forgotten.

First, I am eternally grateful to my advisors Ram and Reza who picked me up, dusted me off, and helped me become the much improved version of myself who I am today. In a similar vein, I am extremely grateful for the warm support, mentorship, and guidance of Walter Willinger whose thinkings and writings have greatly improved my own. I also am grateful for my committee members Zena, Jun, and David, whose insightful conversations challenged me to adopt a wider perspective, and for Arpit and Walt whose collaborative efforts got several of these projects off the ground and across the finish line. Finally, I am grateful for Shahram Davari whose tireless work to sustain our collaboration with Broadcom, Inc. has in no small way enabled several key parts of this dissertation.

I must also acknowledge the previous generation of ONRG students, Bahador and Soheil, who made me feel welcome, accepted, and valued from the day I stepped into the lab; the fellow students who worked with me on various projects along this journey—especially Matt Hall, whose companionship helped make challenging circumstances surrounding the COVID 19 pandemic easier; and, finally, the next generation of ONRG students, Mana, Nima, and Emad, who have vigorously jumped in to start tackling several of the open problems posed by this dissertation.

Beyond the support of mentors and colleagues in computer science and related fields, my work is built on strong emotional and intellectual foundations established by my parents Thomas Misa and Ruth Fothergill. Moreover, none of my present efforts would be possible without the continued support, love, encouragement, and patience of my wife 李陶.

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

Fast and reliable computer networks are a critical service for a wide range of present-day Internet-connected enterprises (*e.g.,* banks, universities, governments etc.). For engineers and administrators tasked with the deployment, expansion, maintenance, and security of such networks, real-time monitoring of packet traffic flowing through the network is a mission-critical capability. In particular, outputs in the form of traffic metrics or events detected from real-time traffic monitoring systems are necessary inputs for network management tasks ranging from detecting and debugging performance issues to attack detection and mitigation to short-term traffic engineering to long-term capacity planning. For example, network traffic monitoring may be used to detect and understand the causes of high latency [123]. Alternatively, network traffic monitoring may be used to detect and mitigate malicious attack traffic to prevent it from impacting benign users [70].

To effectively satisfy the requirements of management tasks in real-world networks, traffic monitoring systems must efficiently process large traffic volumes, remain robust when faced with the complex structure of real traffic, and provide specific, actionable results for a wide range of tasks. Recently, the emergence of programmable switch ASICs [1] that can be configured to compute traffic monitoring results directly in the network hardware substrate raises the promise of network traffic monitoring systems with high packet processing efficiency, low energy and capital costs, and the ability to produce detailed results for a wide range of tasks. However, the high efficiency of programmable switch ASICs necessitates a constrained programming model with access to only a small amount of high-speed memory, a

[1] ASIC stands for Application-Specific Integrated Circuit. See Appendix D for a complete glossary of all abbreviated terms.

limited number of primitive operations per packet, and tens of seconds of network downtime each time the program is changed. As a result, significant research effort investigates designs and algorithms required to realize this promise.

Despite significant effort, particular directions taken by current research on traffic monitoring systems has led to critical limitations in light of real-world traffic structure. Designs and algorithms are proposed based on superficial understanding of what network traffic can be expected to look like in the real world and what particular tasks network management may require. This leads to a pattern where particular design decisions become entrenched and much effort is devoted to making them work for a wider variety of cases rather than questioning their fundamental suitability for particular traffic monitoring problems.

To address current limitations of traffic monitoring system research and to pave the way for principled approaches in future research, we adopt the following.

Thesis Statement. Real-world network traffic is not generated uniformly at random, but exhibits complex statistical structure resulting from human and machine communications. By understanding the statistical structure of network traffic, we realize a novel refocusing of state-of-the-art approaches through proposing and building structure-aware telemetry systems to improve the efficiency and practicality of traffic monitoring tasks.

1.1 Background: Network Traffic Monitoring

Network traffic monitoring involves *(i)* observing packets passing through the network (*e.g.,* as they are transmitted through a router or switch), *(ii)* aggregating packets into groups (*e.g.,* based on source IP address), and *(iii)* computing per-group metrics that enable making group-level decisions (*e.g.,* deciding to block traffic from source addresses associated with a network-based attack). These

three elements are typically combined in a logically-centralized system that allows network administrators to specify the particular groupings and computations for their monitoring tasks and presents the results in a human- or machine-readable format. We refer to such systems as **telemetry systems** and refer to the computations required for particular monitoring tasks as **traffic queries** or **queries** for short. Traffic queries embody particular questions a network administrator may want to ask about traffic flowing through their network and can produce their results based on detection of certain events (*e.g.,* when an attack is detected) or periodic intervals (*e.g.,* every 5 seconds). We envision deployment of telemetry systems on campus or enterprise networks between internal computers and the external Internet with as illustrated in Figure 1. Once deployed, network administrators "ask" the telemetry system queries about their traffic and receive real-time results.

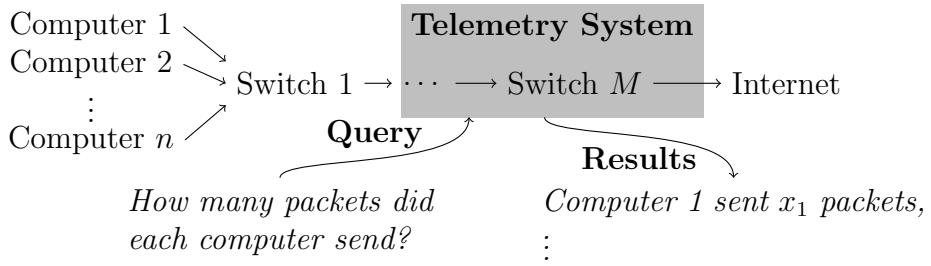


Figure 1. Simple example of a network setting, telemetry system deployment, traffic queries, and results.

The design and implementation of telemetry systems entail *(i)* specification of a query interface that network administrators use to define and submit queries, *(ii)* programming network devices to *observe* network packets as they are forwarded through the network and *compute* per-group metrics, and *(iii)* delivering the results from each group in realtime to interactive dashboards, automated network management systems, or other consumers of information about network traffic. This

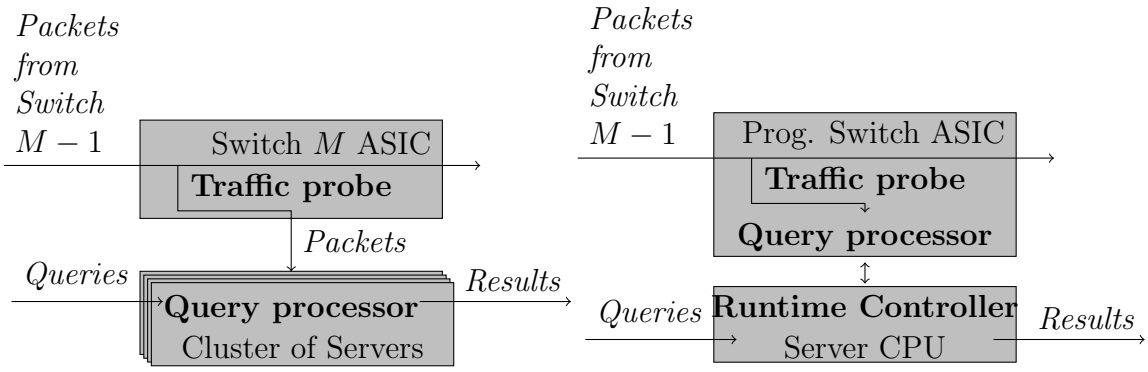
dissertation focuses on *(ii)*, how network devices are programmed to implement telemetry systems, and adopts common models for *(i)* and *(iii)*.

Implementing telemetry systems involves two logically-distinct processes.

- **Traffic probes** observe packets flowing through the network. n
- **Query processors** perform grouping, filtering, and aggregation computations required by traffic queries.

As shown in Figure 2, traffic probes must be "plugged" or "built" in to the network dataplane (*e.g.,* switches forwarding packets between computers) in order to observe live network traffic. Query processors, on the other hand, can either be completely separate from network switches (as shown in Figure 2a), completely integrated into the switch ASIC (as shown in Figure 2b), or somewhere in between with some computations running in the switch ASIC and post-processing computations run on an external server. For example, in most switches the hardware ASIC responsible for forwarding packets also counts the number of packets ingressing and egressing each switch port. This can be understood as executing a simple query that groups packets by switch port and computes the number of packets observed in each group. In other telemetry systems, probes and processors are physically distinct equipment. For example, many intrusion detection systems (IDSs) use the switch hardware ASIC purely as a traffic probe to forward a copy of each packet to a dedicated cluster of query processors (*e.g.,* [129, 20]).

For systems that use separate query processors, a key challenge is handling the high packet rates encountered in real networks. As illustrated in Figure 2a, this challenge often requires distributing the computation across an expensive and power-hungry cluster of servers. Systems that implement query processors directly in the network hardware eliminate this overhead and may only require a single light-weight



(a) Separate traffic probe implemented on switch and external query processor.

(b) Traffic probe and query processor combined in single switch ASIC.

Figure 2. Example of placement of traffic probes and query processors in two different telemetry systems.

server to control query computations and collect the results as illustrated in Figure 2b. We next introduce programmable switch ASICs which are a type of network hardware that can implement some query processor functionalities.

1.2 Background: Programmable Switch Hardware

One of the most important trends in network system design and a key focus of this dissertation is the emergence (around 2013) of "programmable" switch hardware ASICs [40, 8]. Programmable switches (as such ASICs are more colloquially known in the community) extend the match-action model of software-defined networking (SDN) [111] to include user-defined actions that can edit packet headers, perform simple ALU-based computations, and read and write to a limited amount of high-speed SRAM memory *for every packet forwarded through the device*.

As shown in Figure 3, packets enter the programmable switch through input ports (left), pass through the ingress pipeline, queuing stages, and the egress pipeline before being transmitted from output ports (right). Both ingress and egress pipelines typically consist of a parser to extract particular fields from each packet's header,

a number of homogeneous processing stages, and a deparser which reconstructs the packet headers to include any editing performed by the processing stages. Each processing stage performs a set of primitive computations based on table lookups (*e.g.,* using ternary content-addressable memory or TCAM) from packet metadata and values read in from a per-stage SRAM block.

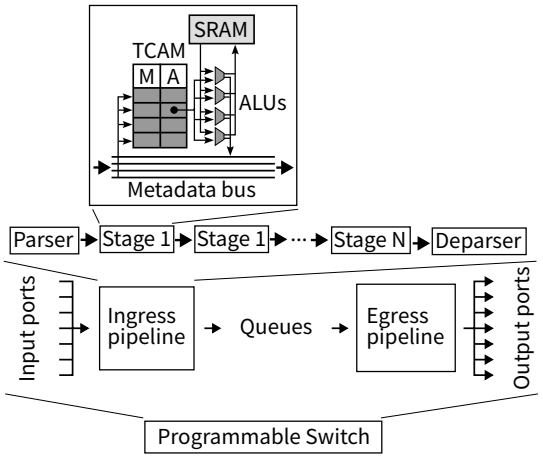


Figure 3. Example packet processing architecture of a modern programmable switch ASIC.

Designers of telemetry systems (or of other network functions) program the switch ASIC in two phases: the **hardware-specification** phase and the **runtime-controller** phase. First, the hardware-specification phase uses a DSL (*e.g.,* P4 [39]) to specify how hardware primitives like TCAM, SRAM, and ALUs will be configured and connected together. In particular, they specify *(i)* how the parsers extract packet header fields, *(ii)* which header fields each (logical) match-action stage can match against, and *(iii)* the particular "actions" or functions which will be used in each stage to modify header fields and SRAM. Actions can also take arbitrary arguments specified in the runtime-controller phase. A hardware-specific compiler then takes this specification and produces two artifacts: *(i)* a low-level configuration that can

be loaded onto the ASIC and *(ii)* a runtime API specification that describes the particular match fields and action parameters that can be manipulated at runtime. Next, the runtime-controller phase uses a general purpose programming language (*e.g.,* Haskell [9]) to dynamically add and remove match-action rules based on the runtime API generated for the particular hardware-specification. For example, to implement a query that counts the number of packets sent from a set of particular source addresses, the hardware-specification could define a single match-action table that matches based on the source address of each packet and then executes a single action that updates a counter associated with that packet's source address in SRAM. The runtime-controller phase could then install rules in this table to match against the particular set of source addresses to monitor and then read the counter values from SRAM for each monitored source address. Note that changes made to the hardware-specification require reloading the switch which leads to tens of seconds of network downtime whereas changes made through the runtime API have no impact on traffic processing. One can imagine (at a high level of abstraction) that the hardware-specification is a compiled function implementation and the runtime-controller invokes that function with different arguments to select different per-packet behaviors.

1.2.1 Benefits. Using programmable switch hardware to implement query processors directly on the switch ASIC (as shown in Figure 2b) drastically improves the performance of telemetry systems. In particular, recent research [123, 70] shows that all or part of the query processor can be implemented directly in switch hardware for a wide variety of traffic monitoring queries (well beyond simple per-port packet counts). This reduces or eliminates the need for sending packets to CPU-based query processors and, in-turn, can lead to orders of magnitude reduction in cost (*e.g.,* capital expense, power utilization) [103] while enabling finer-grain, more detailed traffic

monitoring. Since CPU-based systems are almost always still required to coordinate switch hardware operations, collect intermediate results, and perform any remaining query processing, we refer to such systems as **hybrid telemetry systems.**

1.2.2 Constraints. Programmable switch hardware achieves high efficiency for extremely high traffic rates at the cost of a constrained programming model as summarized below.

- Programmable switches can only make use of a limited amount of high-speed SRAM (*e.g.,* tens of megabytes per pipeline) to compute aggregate metrics for each group. Given physical wiring and thermal constraints it is non-trivial to simply "plug" more memory into the switch ASIC.
- For each packet, programmable switches can only execute limited primitive operations subject to constraints induced by the pipelined computational model shown in Figure 3. In particular, the graph of the computation can have no backward edges, has a maximum width determined by the number of ALUs per stage, and a maximum depth determined by the number of stages in the pipeline.
- Telemetry systems must fix allocation of memory and processing operations, typically when the switch hardware program is compiled. Any aspects of query processing which must be modified at runtime, must be anticipated at compile time and included as runtime parameters in the hardware program definition.[2]

1.3 The Current State of Telemetry System Research

1.3.1 Areas of Investigation. Recent research investigates three distinct areas in relation to the design and implementation of telemetry systems that use programmable switch hardware to implement all or part of the query processor. As

[2]Several recent proposals suggest modification to the underlying switch ASIC hardware architecture that could enable more runtime flexibility [173, 59], but it is unclear if and when such proposals will yield viable hardware products for implementing real-world telemetry systems.

shown in Figure 4, these areas are strongly related in that findings in one area have key technical implications for efforts in one or more other areas. In the following, we precisely define each area and describe connections between areas.

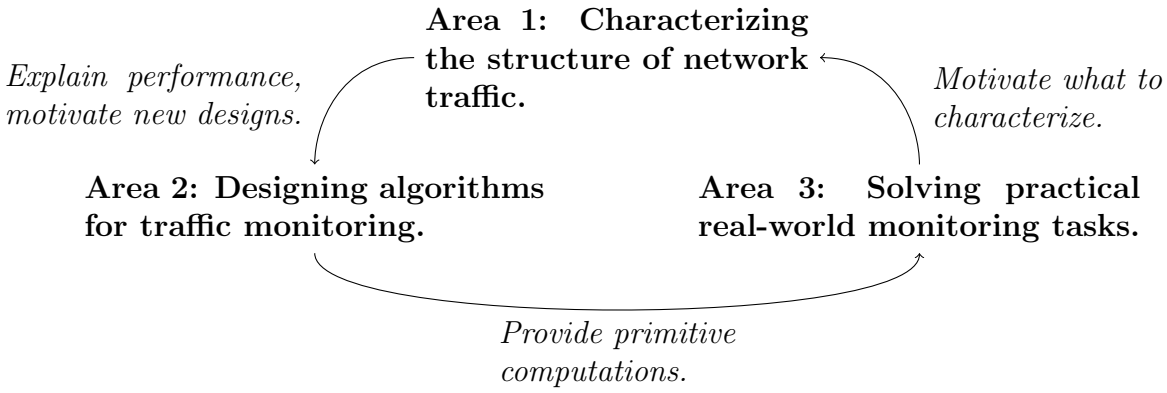


Figure 4. Areas of research related to the challenges of using programmable switch hardware to implement telemetry systems and concrete examples of connections between them.

Area 1: Characterizing the structure of network traffic. Flows of packets observed in network traffic are generated by human activities, such as visiting a website or checking email, as well as by machine processes, such as automatic updates or clock synchronizations. The behavior of these flows (*e.g.,* packet rate) is influenced by the particular path of devices traversed through the network as well as cross traffic from other flows traversing the same devices. Traffic from multiple such flows aggregated at the scale of medium to large enterprises can be understood as a highly-structured stochastic process with distinct statistical patterns and distributions. Understanding these patterns and distributions is a foundational requirement for design and operation of many network components including telemetry systems. In particular, due to limited memory available on programmable switch hardware, for telemetry systems the key structural features of concern are how many traffic

groups (*e.g.,* IP addresses) are observed over time and how the keys of these groups are distributed in the key space (*e.g.,* the space of all possible IP addresses). In considering this connection a key open questions is *what particular aspects of traffic structure are important to characterize and how can state-of-the-art statistical tools be used for such characterizations and understood in networking terms?*

Area 2: Designing algorithms for traffic monitoring. Telemetry system design requires careful selection and development of algorithms to efficiently compute query results over large volumes of traffic in real-time. When leveraging programmable switch hardware in particular, the designs and algorithms chosen to implement traffic queries must be mapped into the constrained programming or configuration model described above. On the one hand, statistical characterizations of traffic structure (*e.g.,* the number of expected groups observed per time window) directly inform algorithm design decisions. On the other hand, designs and algorithms for telemetry systems are only useful insofar as they are able to effectively apply to relevant traffic monitoring tasks on real-world traffic. When considering these connections, a critical open question is *how to come up with designs and algorithms that leverage traffic structure and support practical real-world monitoring requirements?*

Area 3: Solving practical real-world monitoring tasks. Network administrators, at the end of the day, care about high-level questions about network traffic to drive network management decisions. In many cases it is non-trivial to map such high-level questions from their semantically ambiguous representation in human language into concrete traffic queries as defined above. Multiple different queries could all be construed to produce results for a single high-level task or a single high-level task could require multiple queries and careful combination of query results. For example, suppose a network administrator wants to know when a volumetric DDoS attack

might be happening on their network. Depending on the types of attacks launched by adversaries and the baseline traffic profile of the defending network, either a simple query that reports when the total aggregate traffic volume exceeds a threshold could suffice or more complex queries based on distinct counts [70] or entropy [52] could be required. When considering these connections, a critical open problem is *how to refine real-world network management tasks into concrete well-defined traffic queries?*

1.3.2 Limitations of Prior Efforts. Three critical limitations in prior telemetry system research efforts can be understood as failure to fully consider the deep connections between the areas outlined above.

Limitation 1: *characterizations of traffic structure lack insights into traffic features relevant to real-world tasks* (Area 3 $\leftrightarrow$ Area 1). Most practical tasks involve detecting particular IP addresses (*e.g.,* addresses associated with network-based attacks or particular performance problems), but the structure of observed IP addresses has limited formal characterization. As a result, algorithms for detecting particular IP addresses start from excessively general assumptions (*i.e.,* that they must work for any arbitrary distribution of observed IP addresses) and miss opportunities for optimization based on structural knowledge. Efforts to characterize traffic structure, on the other hand, have focused on time-domain metrics like packet and byte rates, but do not provide detailed or high-confidence characterization of metrics like how many addresses are observed at a given time and what are their spatial distributional structure.

Limitation 2: *query processor algorithms are not informed by real-world traffic structure* (Area 1 $\leftrightarrow$ Area 2). Most state-of-the-art traffic monitoring algorithms rely on sketch-based approximation methods [50, 178], but the accuracy of these methods results from interaction between the amount of memory allocated

and the number of distinct groups (*e.g.,* IP addresses) observed. On the one hand, as mentioned above, the amount of memory available on programmable switch hardware is small and fixed. On the other hand, findings from **Area 1** firmly establish that traffic rate time-series are self-similar or highly bursty over a wide range of time scales [97, 34, 22, 171]—a finding which our efforts in **Area 1** extend to the number of observed groups per time unit. As a result, such approximation methods are highly limited in their ability to maintain reasonable accuracy through the normal fluctuations in number of observed groups.

Limitation 3: ***proposals for real-world solutions inherit limitations of existing algorithms*** **(Area 2 $\leftrightarrow$ Area 3).** Practical system proposals often use heuristic-based methods built atop simple traffic metrics and do not consider designs and algorithms that could enable richer, more effective sets of metrics. As a result, research proposals for particular monitoring tasks focus on optimizing efficiency of particular well-known algorithms (*e.g.,* sketches), but are limited in their practical utility when faced with real-world task and traffic workloads.

1.4 Organization of This Dissertation: Focusing on the Connections

In order to firmly identify the above limitations in existing research efforts, this dissertation first reviews prior work in Chapter II. In § 2.1 we review prior characterizations of traffic structure (**Area 1**) and describe how they fall short of providing solid foundations for the design of telemetry systems. In § 2.2 we review existing designs and algorithms used in state-of-the-art telemetry system proposals (**Area 2**) and describe how they are limited in practice by lack of consideration of traffic structure. In § 2.3 we review proposed uses of traffic queries to answer practical real-world questions about traffic (**Area 3**) and discuss how these uses are limited

by a lack of insight into how the designs and algorithms they adopt interact with the structure of the traffic they consider.

In light of these limitations of prior work, we present focused research contributions that leverage connections between areas as illustrated in Figure 5 and described in the following. Since these connections have already been hinted at in prior work, our primary contribution is not the establishment of this connective structure per se, but the demonstration of how deeper exploitation of these connections can lead to new advances in each particular research area.

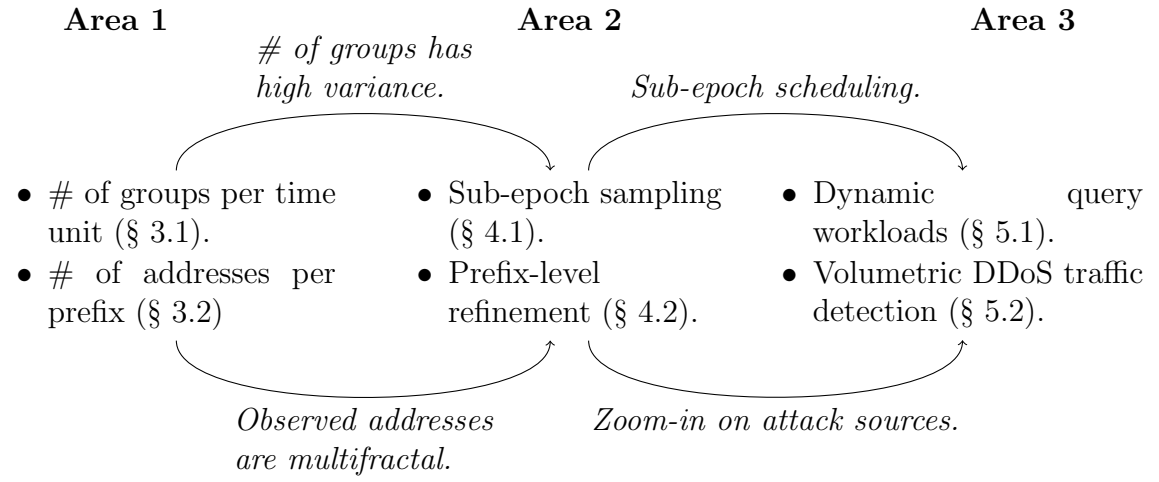


Figure 5. Structure of the principled approach taken in this dissertation highlighting the connections exploited between the areas of Figure 4.

- In Chapter III we present foundational characterizations of specific aspects of traffic structure that are relevant to telemetry system design. We show that the number of traffic groups observed has high time-domain variance, but falls short of full self-similarity (§ 3.1) and that Internet addresses (which group packets in address space) follow multifractal scaling (§ 3.2). The contributions in this chapter address **Limitation 1** by characterizing aspects of traffic relevant to real-world traffic monitoring requirements and providing key insights to the next chapter.

- In Chapter IV we present designs and algorithms that leverage our characterizations of traffic structure. We develop a sub-time-window (or sub-epoch) approach to sampling and scheduling traffic query computations which exhibits better robustness in the face of high-variance time structure (§ 4.1) and algorithms for prefix-level iterative refinement which reduce required switch hardware memory by exploiting the multifractal spatial structure (§ 4.2). The contributions in this chapter address **Limitation 2** by developing designs and algorithms that are informed by characterization of relevant aspects of traffic structure.
- In Chapter V we connect our designs and algorithms to practical traffic monitoring tasks and evaluate their performance under realistically-structured traffic workloads. We develop a system called DynATOS [118] that answers a large and dynamic set of per-group time-series queries (§ 5.1) and a system called ZAPDOS [117] that efficiently detects the sources of volumetric DDoS attack traffic (§ 5.2). The contributions in this chapter address **Limitation 3** by implementing and evaluating practical telemetry systems for real-world traffic monitoring tasks.

Finally, we conclude in Chapter VI by considering potential for future work to further strengthen the connections between the three research areas.

1.5 Previously Published and Co-Authorship Statement

Content in Chapter II is intended to be published with co-authors Reza Rejaie and Ramakrishnan Durairajan. Content in Chapter III, Appendix A, and Appendix B is intended to be published with co-authors Reza Rejaie, Ramakrishnan Durairajan, Arpit Gupta, and Walter Willinger. Content in Chapter III, Chapter IV, Chapter V, and Appendix C is published with co-authors Walt O'Connor, Reza Rejaie, Ramakrishnan Durairajan, Arpit Gupta, and Walter Willinger. To clearly

separate from technical content, details of publication status and co-authorship are denoted in an *italic font* at the beginning of each chapter or appendix.

CHAPTER II

PRIOR WORK

The organization of prior work described in § 2.2 and § 2.3 was developed in collaboration with Ramakrishnan Durairajan and Reza Rejaie. I did the primary work of reading all papers, catagorization of papers, and composition of text.

This chapter describes the three research areas identified in Chapter I by establishing a detailed internal structure of each area and reviewing the findings of relevant research within these internal structures. In particular, we review work on traffic structure characterization in § 2.1, work on designs and algorithms in § 2.2, and work applying telemetry system designs and algorithms to real-world traffic monitoring problems in § 2.3.

2.1 Characterizations of Traffic Structure

We first discuss prior efforts to characterize the structure of network traffic (**Area 1** from Figure 4). Though there are many different aspects of traffic to characterize, in this work focus on two aspects with particular relevance for design and implementation of telemetry systems. In particular, § 2.1.1 discusses work on characterizing time-domain aspects and § 2.1.2 discusses work on characterizing address-space-domain aspects. Because the particular aspects of traffic structure relevant to the design and implementation of telemetry systems are less well studied in recent work, we take a less structured approach than in later sections of this chapter.

2.1.1 Temporal Structure Characterizations. Historically, characterizations of the temporal structure of network traffic focuses on aggregate traffic rates (*e.g.,* total number of packets transmitted or total number of new flows per time unit). Accordingly, raw traffic data can be summarized in the form of a time series

(or "signal") $X = (X_t : t = 0, 1, 2, \ldots)$ where for example $X(t)$ denotes the number of packets or bytes observed in the measured traffic during the t-th time interval and statistical analysis tools used for time series in other domains are applied. A long line of work in the late 1990s seeking to identify the correct analysis tool and to justify that tool in terms of networking-specific concepts led to the high-level understanding that traffic rate time series are asymptotically self-similar or mono-fractal [97, 34, 22, 171]. Intuitively, this means that these time series are highly irregular or "bursty" regardless of their time-domain resolution or aggregation level.

The foundational characterizations established by these works were motivated by, and ultimately lent critical insights to, a key problem in networking at the time of determining the *optimal* queue length. Queue length has a significant impact on performance in real network settings and an optimal queue length would prevent excessive packet loss (from too short of a queue) and excessive latency (from too long of a queue) [57]. However, the eventual confirmation of asymptotic self-similarity in observed time series implies that there is no optimal queue length (because there may always be a larger burst) [97] and led to a refocusing of efforts around congestion management. Such efforts continue in more recent efforts to leverage capabilities of programmable switches for automated queue management [93, 68].

What time series are relevant for telemetry system design? For telemetry systems implemented using programmable switches, this time-domain structure of aggregate traffic rate is not critical because query processing is executed deterministically at high-rate in the switch's ASIC. In particular, programmable switch ASICs typically use a pipe-lined processing model with a fixed clock provisioned to achieve aggregate throughput near or exceeding the combined throughput of all ports into and out of the switch. Packets are only lost when queues

associated with egress ports become full and hence such burst-induced packet loss has minimal impact on query processor functions which are typically implemented on the main (high throughput, deterministic) ingress or egress pipelines.

Unfortunately, the key aspect of time-domain structure that has significant bearing on the design of telemetry systems—in particular, the **number of traffic groups observed per time unit**—has received less direct research attention. To understand the significance of this particular aspect let $X'(t)$ denote the number of distinct source addresses observed during the t-th time interval and consider a traffic query that counts the number of packets sent from each source address. It is easy to see that the amount of memory required to compute this query exactly in the t-th time interval is directly proportional to $X'(t)$. As a result, the dynamics of how $X'(t)$ evolves over time directly impact the question of how much memory the query needs to be allocated. If more sources are observed than fit in the memory allocated, the query's results will be inaccurate. Note that this particular aspect is subtly different from the flow arrival rate considered in some prior work [74, 76] because it is a result not only of when a particular flow arrives (or a particular source is first observed), but also of how long that particular flow remains active.

The closest prior work comes to characterizations of "number of groups per time unit" is through constructive approaches that model network traffic as the sum of an underlying flow arrival process [75, 125, 73]. Such works form a core component of the explanation for why observed data rates are self-similar (and not better described by some other, *e.g.,* multifractal model) by connecting the observed statistical behavior to well-known facts about how networks operate (*i.e.,* packets are transmitted in groups associated with flows). However, they also establish that the underlying flow-arrival process does not need to have self-similar structure. For example, the authors

of [75] demonstrate that self-similarity of data rates is not dependent on a particular time-domain structure of flow arrivals and can be replicated realistically using a simple Poisson flow arrival process. Given the close relationship between flow arrivals and arrival of new groups for a telemetry system to monitor, this initial result suggests that the time-domain structure of the number of groups is perhaps less variable, and hence expected to have less impact on telemetry system operations, compared to the parallel line of work investigating data rates and queue management.

The resulting implication for telemetry system research thus far is that researchers assume the number of groups per time unit will remain relatively constant over time and little effort is invested in fully characterizing this aspect of temporal structure. In § 3.1 we will challenge this notion, demonstrating that in real network traffic the number of groups per time unit is a dynamic, high-variance metric, and in § 4.1 we develop algorithms for query processors that remain more robust compared to state-of-the-art in the face of this variance.

2.1.2 Spatial Structure Characterizations. Characterizations of the spatial structure of traffic focus on understanding the locations of the computers generating the traffic and how these locations are distributed in different "spaces". These spaces include physical space (in which location corresponds to geographic location) [92], topological space (in which case location corresponds to position in the network graph) [30], or address space (in which case location corresponds to a particular address used to identify the computer in a particular communication protocol) [77]. Of course locations in different types of spaces are not independent and often ranges of locations in one space corresponds to ranges of locations in another space. Consider for example regional Regional Internet Registries which

are associated with networks in a particular geographic regions and assign addresses from particular contiguous blocks to these networks.

This dissertation focuses in particular on structure of the address space because *(i)* programmable switches directly observe addresses in the headers of packets they forward, *(ii)* many telemetry queries use addresses to separate packets into groups associated with particular computers, clients, or attackers, and *(iii)* addresses typically have strong correlation with location in the other spaces (though inferring these correlations is often non-trivial and outside the scope of this dissertation). Formally, locations in the address space are 32- or 128-bit integers (for IPv4 or IPv6, the two currently used versions of the Internet Protocol, respectively) and are organized in a hierarchy based on common prefix bits. For an observed set of addresses (*e.g.,* all addresses observed by a switch during the t-th time interval), we quantify its spatial structure by considering the measure μ_A that assigns to the subset defined by each prefix the number of observed unique addresses that fall in that subset.

Characterization of the spatial structure of μ_A is important for telemetry system design because for common queries which group traffic based on address, it corresponds directly to the structure of group identifiers or *keys* which must be tracked and reported with query results. For example, suppose it can be shown that all addresses observed in a particular network by a particular query will have distinct 8-bit prefixes. In this case, the telemetry system can simply observe the first 8 bits of the addresses in each packets header and allocate $2^8 = 256$ memory entries to keep track of all possible groups.

A brief cluster of work in the early 2000s [90, 91, 31] were the first to statistically analyze the spatial structure of addresses in uni-directional measured

network traffic, providing preliminary evidence that this structure is multifractal in nature. Intuitively, μ_A is said to be a *multifractal measure* (multifractal, for short) if it exhibits a scale-invariance property in the address space whereby clusters of observed IP addresses at any spatial scale (*e.g.,* /16 subnet granularity) tend to fragment into sub-clusters at finer scales (*e.g.,* /17 or finer subnet granularity). It is this qualitative "cluster-within-cluster" or "nesting of structure" property that results in the absence of a "typical" size of a cluster and causes the observed IP addresses to be very unevenly or highly intermittently distributed across the overall IPv4 space.

Despite being published ∼20 years ago, a lack of clearly articulated reasons for why such structure matters has led to limited progress in establishing a rigorous foundational basis for this reported evidence. This lack of progress and associated lack of awareness in the research community in turn engendered wide proliferation of techniques that are address space agnostic in state-of-the-art telemetry system design. As such, the community investigating applied designs and algorithms miss opportunities to leverage this structure in their designs and the community investigation traffic structure is un-motivated to complete the remaining effort required to establish multifractal scaling of observed addresses at a similar depth of certainty as has been established for the self-similar behavior of traffic rate time series.

In § 3.2 we present an initial effort towards establishing the foundationality of multifractal scaling based on examination of address allocation policies and in § 4.2 we will demonstrate how novel algorithm designs can leverage multifractal scaling to improve performance of certain types of telemetry queries.

2.2 Designs and Algorithms for Traffic Monitoring

Next, we describe prior efforts in designing algorithms for traffic monitoring (**Area 2** in Figure 4). Work in this area can be further divided between efforts that use sketch-based algorithms for approximate query processing (§ 2.2.1) and work that uses a stream-processing model for exact query processing (§ 2.2.2).

2.2.1 Sketches. Sketch-based algorithms (or "sketches" for short) refer to a family of approximate algorithms for computing aggregate metrics over a stream of data elements [50, 46, 114]. In the case of telemetry systems, the data elements are typically packets and the metrics are typically count-based summaries like (approximate) number of packets per flow, total entropy over all flows, or a list of "heavy" flows.

Sketches are built using the same primitive operations as hash tables—computing a hash value for a given key and using this hash value as an index into an array of counters. Rather than avoiding hash collisions by using linked-list buckets, probing, or other methods like Cuckoo hashing [127] which all require non-constant operations per update, sketches embrace hash collisions and find creative ways to deal with the consequences. The canonical example, known as count-min sketch [50], updates several hash tables with pair-wise independent hash functions in parallel and takes the minimum value across all hash tables as the estimate of a particular key. The core intuition behind why sketches work is that if the hash functions are (near) independent, then the probability that all hash functions collide for the same pair of keys decreases (near) exponentially in the number of hash functions used.

2.2.1.1 Single (Fixed) Aggregation Granularity. We first consider the simplest case of sketch algorithms designed for implementation on programmable switch hardware that compute a single metric for a fixed grouping of packets.

Common design patterns. These works all have some high-speed, hash-based update method which is compiled into switch hardware to produce tables of counters for a fixed granularity and a fixed metric. These counters are periodically pulled to a CPU-based collector which performs additional statistical procedures on counter values to produce the final estimated traffic profile metrics.

A key challenge in this design space is the fact that sketch algorithms don't store the observed flow keys by default. In particular, the classic solution of maintaining a heap or other tree-like set data structure used in other applications of sketch algorithms (*e.g.,* in databases [66, 72]) requires non-constant operations per update and is hence not possible in switch hardware. The works in this section develop novel solutions to this problem when addressing metrics that require flow keys such as heavy hitters, or define metrics over all flow keys such as entropy.

Key contributions. Perhaps the first solution to compute a traffic profiling metric entirely in switch hardware, in particular heavy-hitters, is HashPipe [147]. HashPipe starts with a well-known algorithm to approximately computes the top-n elements in a data stream [113], then makes several modifications and relaxations of the original algorithm—in particular, sampling to estimate the minimum value and pipelining to fit the minimum estimation into a fixed number of switch hardware stages. Several subsequent works further improve heavy hitter detection on switch hardware to coordinate detection of network-wide heavy hitters by dynamically adjusting thresholds at each switch [71] and to use limited recirculation of packets [33] to fix the critical inefficiency in HashPipe of always evicting an entry (even for flows with a single packet).

Whereas HashPipe and the works mentioned above focus on packet or byte count heavy hitters, SpreadSketch [157] focuses on heaviness in terms of distinct entities

associated with each flow (*e.g.,* distinct sources contacted by each destination), adopting a similar approach of maintaining candidate flow keys associated with sketch counters. Snappy [47] also extends the notion of packet or byte count heavy hitters to include subtraction in order to identify which flows make the largest contribution to long queues. However, Snappy does not maintain flow keys in switch memory, but acts directly on packets given the current estimate of the packet's flow's rate from the sketch counters.

Beyond variants of heavy-hitters, several other metrics have been computed in switch hardware. Following a similar methodology as HashPipe, QPipe [83] uses a near-optimal existing algorithm called SweepKLL [82] for computing quantiles. However, instead of modifying or relaxing SweepKLL, QPipe instead develops a novel technique of using unsampled packets as "workers" to facilitate the required arg-min computation between stages. A similar methodology is followed in [94] to implement a near-optimal entropy-estimation algorithm using a look-up approximation to the required maximally skewed alpha-stable distribution. Dart [142], on the other hand, develops a new approach using heuristics and lazy cache eviction to decide which SEQ packets in one direction of a TCP flow are most likely to produce valuable RTT estimates when paired with their corresponding ACK packets from the opposite direction.

Open problems. The first clear open problem is that of coverage: there are many more traffic profiling metrics (*e.g.,* bandwidth estimation [49, 62, 55]) that could be useful even when computed in the limited single-aggregation form assumed in this section.

The second open problem is a less obvious consequence of the prevalent use of sketch-based techniques in these works. As discussed in § 1.3.2 (**Limitation 1**), while sketch-based techniques do have rigorously-defined statistical error guarantees [50,

46], the accuracy of most sketches actually is a function of the total number of distinct keys tracked. In order to both decide on an appropriate memory allocation for particular sketches as well as to interpret the accuracy of retrieved sketch counters, network administrators first need to know the ground truth number of keys collected in the sketch. This leads to critical problems in real network settings where the number of distinct keys (*e.g.,* the number of currently active flows) changes rapidly and is hard to measure directly. Sketch-based approaches must be made more robust against changes in the number of keys, or that other approximation methods must be considered.

2.2.1.2 Multiple Aggregation Granularities. In several cases, network administrators need to compute the same metrics over different sets of traffic and/or at different granularities. For example traffic could be aggregated both by source addresses and (independently) by destination addresses in the same system whereas single aggregation granularity systems would require one instance for aggregating source addresses and a separate, independent instance for aggregating destination addresses. Here we consider works that address this need head on by computing single, fixed traffic profiling metrics, but exposing interfaces to allow collecting these metrics over different traffic slices and aggregations.

Common design patterns. The works in this section define a limited universe of possibilities which can be understood as a global key space. Individual monitoring tasks or queries all compute the same metrics, but can be parameterized by different subsets as well as different aggregation granularities within this global key space. The key challenge faced in these works is how to design, implement, and analyze data structures that can collect profiling data for multiple aggregation

granularities simultaneously with greater efficiency that simply capturing each granularity independently (*e.g.,* with methods from § 2.2.1.1).

Key contributions. Beaucoup [48] makes perhaps the first contribution in this space by developing an approach to answer multiple threshold-based heavy-distinct-count queries where each query has a different aggregation granularity. For example, Beaucoup can simultaneously monitor for source addresses that contact more than a given number of destinations as well as destination addresses that contact more than a given number of sources. The key enabling idea is to interpret the heavy-distinct-count metric computation as a coupon-collector problem where each packet is associated with a hash-based "coupon" and each query, key pair is a coupon-collector. Given a fixed number of coupons and query, key pairs, Beaucoup then describes a simple mapping of the coupon collector problem into programmable switch hardware.

CocoSketch [182], on the other hand, addresses the problem of computing per-flow packet and byte counts for a flexible range of flow definitions (*e.g.,* five-tuple-based flows, source-based flows, etc.). The key idea is to map the per-flow counting problem into the subset-sum estimation problem and apply the Unbiased SpaceSaving algorithm [159]. As with HashPipe [147], CocoSketch also applies modifications and relaxations to remove circular dependencies in the Unbiased SpaceSaving algorithms making it computable in switch hardware.

Open problems. Similar to fixed-granularity approaches discussed in § 2.2.1.1, a fundamental open problem in these works is how to apply variable granularity approaches to more types of metrics. The relative sparsity of this section is a testament to the fact that this is a significantly harder challenge compared to computing traffic profiling metrics at fixed granularity.

Also similar to the fixed-granularity approaches, the works in this section compile fixed-sized data structures (*e.g.,* for a given fixed number of coupon-collectors in Beaucoup) into programmable switch hardware while the underlying number of monitored keys inherently varies leading to resource allocation and accuracy estimation challenges. For works supporting multiple aggregation granularities, this problem is even more challenging to address because each monitored aggregation granularity may have different numbers of distinct aggregate groups (*e.g.,* a different number of sources and destinations) so allocation must be made per aggregation granularity.

2.2.1.3 Select Metrics From Post-Processing of Fixed Dataplane Results.

Several works leverage the observation that in addition to per-flow packet and/or byte count estimates, sketch counters collected from switch hardware can also be used to estimate several higher-level metrics through CPU-based post processing. These works enable network administrators to compute a fixed set of a few traffic profiling metrics simultaneously through the same switch hardware computations.

Common design patterns. The works in this section develop single sketch-based data structures that are updated directly in switch hardware, then post process the resulting sketch counters on CPU-based systems to estimate a fixed set of metrics typically including heavy hitters, flow-size distribution summaries, entropy estimates, and cardinality estimates (*e.g.,* the number of distinct flows). Note that all of these estimates are limited to the aggregation granularity at which the data plane sketch is compiled. For example, if the sketch is compiled per-destination address, then post processing can extract heavy destination addresses, entropy of estimations, or the number of distinct destinations, but cannot extract metrics w.r.t. other aggregations (*e.g.,* w.r.t. source addresses).

Key contributions. UnivMon [102] heralded the idea of being able to produce multiple metrics through post-processing a single simple data structure through the idea of "universal sketching". Universal sketching builds on streaming algorithms research [41, 42] and essentially allows estimating a class of metrics that can be expressed as sums of a function of the frequencies of individual elements in the data stream (with the additional requirement that these functions must be upper-bounded by the second frequency moment). In practice, the set of metrics computable through this method corresponds to heavy-hitter-type metrics as well as global metrics like cardinality and entropy. Note that several further optimizations of the switch hardware implementation of UnivMon are discussed in [103, 104, 172].

Despite the novel theoretic underpinning, UnivMon has several limitations which were picked up in subsequent works. SketchLearn [78] develops a method to automatically extract flow keys from a multi-level sketch using statistical modeling techniques. ElasticSketch [176] explicitly deals with changes in the number of keys and distribution of traffic across keys commonly observed in network traffic by splitting the sketch data structure into heavy and light parts with different update policies. Most recently FCMSketch [151] and [79] develop methods for further improving the accuracy of similar sketch-based approaches using a hierarchical tree-like organization of sketch counters and a correspondence with compressive sensing respectively.

Another approach taken in StarFlow [150] and TurboFlow [149] takes the late-binding principle farther, using switch hardware only to group packet-level data into aggregates (*e.g.,* flows), then exporting succinct summaries of all packets in each aggregate (*e.g.,* StarFlow's "grouped packet vectors"). Although this approach enables a wider variety of metrics compared to sketch-based works, it also imposes

higher load on the CPU-based post processing system and makes limited use of switch hardware's aggregation capabilities.

Open problems. As with the single, fixed aggregation solutions described in § 2.2.1.1, the works in this section all statically fix a particular aggregation granularity in the switch hardware program. In many ways this limits the universality of proposals like UnivMon [102] since separate independent instances must be maintained for all aggregation groups network administrators wish to profile. In a similar way, the use of sketches in may of these works is still plagued by the fundamental connection between sketch accuracy and the variable and unpredictable number of flows observed in network traffic. Importantly, although ElasticSketch [176] appears to address this issue head-on, their solution is still somewhat brittle (using a strict two-tier hierarchy) and focuses more on adapting to changes in traffic rate in software versions of the same algorithms.

2.2.1.4 Select Metrics From Fixed Menu.

The works in this section allow network administrators to construct custom traffic monitoring switch hardware programs by selecting different types of metrics (*e.g.,* heavy hitters, cardinality) from a fixed menu and parameterizing each selection to look at particular slices of traffic and at particular aggregation granularities.

Common design patterns. These works start by defining a set of metrics typically implemented through sketch-based algorithms then define methods for combining multiple algorithms in a single switch hardware program. The key challenge faced in these approaches is how to minimize the switch hardware resources required to implement a given set of metrics or sketches. Although sketches on the surface seem to perform essentially the same types of computations and should be trivial to

merge, when differing traffic slices, keys, and sketch algorithms are involved, creative combination strategies are required.

Key contributions. OpenSketch [178] demonstrated the first such system defining a flexible FPGA-based data plan that performs hashing, TCAM-based classification, and counting primitives and a library-based control plane containing preset formula for how to configure the data plan primitives for a variety of traffic profiling metrics. OpenSketch also proposed an active "sketch manager" which performs additional traffic profiling to tune memory allocations of the currently running profiling queries. For example, the sketch manager might run distinct-count profiling to automatically adjust the number of counters assigned to a heavy hitters query as the underlying number of keys changes.

Several recent works including HeteroSketch [23] and SketchLib [122] propose similar methods for coordinating execution of multiple sketch-based traffic profiling queries in a common system and for managing their resource allocations. In addition to targeting the modern generation of P4-programmable switches (instead of requiring an FPGA) and incorporating support for sketch algorithms adopted after OpenSketch (*e.g.,* UnivMon [102]), these works also make several targeted resource usage improvements [122] and propose a unified framework for understanding the processing performance of sketch algorithms on diverse hardware and software targets [23].

Open problems. As with other traffic profiling works, there is a continuous open problem of incorporating support for more diverse metrics into the the "fixed menus" of works in this category. Also, although OpenSketch [178] proposed an approach to dynamically adjust sketch counter allocation in response to changing numbers of flows, this approach requires extra overhead and does not address technical challenges

faced when deploying such an approach on programmable switches. More recent works [23, 122] do not provide adequate solutions to the general challenges with provisions sketch counters and estimating accuracy of sketch-based results.

2.2.1.5 Takeaway. Although sketch-based algorithm design can be adapted to implement several different types of queries, it is fundamentally limited by consideration of the real-wold high-variance structure of the number of groups per time unit (*i.e.,* connection between **Area 1** and **Area 2**). This high-variance structure combined with the relationship between number of groups and sketch accuracy implies that sketch-based approaches will always fall into one of two extremes: being over-provisioned for the actual number of groups and wasting SRAM or being under-provisioned for the actual number of groups and suffering accuracy degradation.

2.2.2 Stream-Processing Framework. Another approach to designing algorithms for telemetry systems that collect multiple metrics is to allow network administrators to specify the metrics they seek to collect by combining a fixed set of primitive operations and combinators in a domain specific language. Rather than providing a fixed menu of metrics, the works in this section develop more flexible primitive operations, often specified in a stream-processing-like language, and define algorithms for how these primitives can be composed on switch hardware. Note that such stream-processing queries are a superset of the traffic queries defined in Chapter I because they can have multiple, pipelined grouping and aggregation stages.

2.2.2.1 Static Traffic Queries. The first type of work accepts a list of stream processing queries when the system is initially deployed. During the systems deployment, results from these queries are returned to network administrators, but the queries cannot be changed without reloading the system, which also induced tens of seconds of network down time.

Common design patterns. The primary technique in this type of work is to compile query specifications from a stream-processing language (*e.g.,* Apache Flink [1]) to a switch hardware program, the associated control-plan scripts, and a program to complete any post-processing for operations that cannot be computed in switch hardware. The key challenges then are *(i)* how to define these languages to be expressive enough to allow useful metric computations but limited enough to still be compilable to the limited switch hardware processing model and *(ii)* given such a language and compiling method, how to optimize hardware resource usage in the face of unknown and unpredictable traffic loads.

Key contributions. Marple [123] initiated the DSL-based approach considered in this group by proposing "language-directed hardware design". In particular, Marple defines a stream-processing language with functional "map", "filter", "group-by", and "zip" primitives and describes how each of these primitives can be mapped into switch hardware programs (or in the case of "group-by" a combined hardware, software construct). Sonata [70] extends Marple by partitioning the language primitives of given queries between the switch hardware program and a CPU-based post-processor, thus breaking the strict limits on query complexity imposed by switch hardware limits in Marple. Concerto [99], DynamiQ [36], and DynaMap [146] further build on Sonata adding support for deploying queries across distributed programmable switches and enhanced supported for automatically adjusting resource allocations in response to changes in traffic composition respectively. EQuery [132] also proposes an expressive event-based language which can be compiled to efficient non-deterministic finite automata (NFA), but does not appear to poses a concrete switch hardware implementation.

Open problems. Although the works in the section follow a programming-language-based approach to network traffic profiling, they connect only superficially to the rich literature on programming language theory. In particular, providing formal semantics as well as investigating other compiler-level optimization techniques remain open problems. Given the recent interest in formalizing low-level switch programming interfaces [88, 53, 139, 25], this may be a promising direction for future work.

Additionally, since aggregation operators are still implemented with sketch-based primitives (*e.g.,* Sonata's [70] "reduce" operator), these works also face issues with provisioning and error estimation. While works like DynamiQ [36] make some progress in this direction, they still focus on optimizing processing performance (*e.g.,* reducing the volume of tuples exported to the CPU-based collector) and leave result accuracy as a secondary concern. The flexibility provided by these language-based approaches further complicates sketch error estimation since reduction operations can be performed on arbitrarily filtered substreams and/or pipelined for multiple levels of aggregation.

2.2.2.2 Support Runtime Changes.

Whereas works in the previous categories focus on statically configuring switch hardware to collect particular sets of metrics, another type of work seeks to enable network administrators to dynamically add and remove metrics or queries during runtime.

Common design patterns. In this approach, switch hardware acts more as an interpreter which is dynamically configured by a control plan to execute different aggregation computations over time. In particular, the programs compiled into switch hardware employ an extra level of indirection to allow switching between different computations based on the values of different ASIC "control" registers. A

software controller can then control the per-packet computation without reinstalling the hardware program by simply writing different values to the control registers.

These works face similar challenges in balancing expressiveness of their query interfaces with feasibility of implementation through register-controlled computations. Additionally, whereas in previous approaches switch resources are statically allocated at compile time, works in this section face a dynamic resource allocation and/or scheduling challenge.

Key contributions. Initial efforts in DREAM [120] and SCREAM [121] leveraged fixed sets of TCAM-based counters in the previous generation of SDN-programmed switches to allow runtime control of multiple parallel instances of three types of queries (heavy hitters, hierarchical heavy hitters, and change detection) parameterized by traffic slice and aggregation granularity. In addition to supporting addition and removal of queries, these works actively adjust resource allocation among all running queries based on accuracy signals derived for the three particular query types.

More recently, works like Newton [187] and Flymon [184] developed methods for dynamically adding and removing arbitrary queries written in Sonata-like [70] stream processing languages from switch hardware during runtime. The key contribution of these works is the design of switch hardware programs that bake enough flexibility into the hardware pipeline so as to allow a network administrator to execute arbitrary stream processing queries simply by runtime configuration. DynATOS [118] further extends these works by developing a novel approximation and resource scheduling approach to execute the first aggregation stage in such runtime-controlled query systems while adapting to changes in query work load as well as changes in the underlying network traffic.

Open problems. A key open problem in this research direction is how to implement more generic approximation and resource management techniques that can adapt to complex query definitions with multiple levels of aggregation (note that the methods described in DynATOS [118] only apply to the first level of aggregation regardless of the complexity of the target query). This challenge also relates to the previously mentioned challenges with sketches: either methods to dynamically adjust sketch counter allocations must be developed, or entirely new approximation approaches must be explored (*e.g.,* based on sampling theory as proposed in DynATOS [118]).

2.2.2.3 Takeaway. Although stream-processing-based design approaches enable a large number of different traffic queries, it is unclear how useful these types of queries will be for real-world network traffic monitoring needs (*i.e.,* connection between **Area 3** and **Area 2**). In particular, these works inherit the limitations of the sketch-based primitives on which they are often based without considering how the surrounding stream-processing operators (*e.g.,* different filters, different grouping definitions, pipe-lining of primitive operations) may impact sketch accuracy. Moreover, fine-grained traffic monitoring tasks required to solve real-world networking problems often require complex order-dependent aggregation metrics in each group (*e.g.,* duration between the TCP three-way handshake and connection teardown) which cannot be easily computed given the limited fixed set of (typically commutative) metrics considered in state-of-the-art stream-processing-based efforts.

2.3 Applications to Real-World Tasks

Prior work providing concrete application of the capabilities of telemetry systems to real-world traffic monitoring tasks typically consider tasks that seek to understand performance issues (described in § 2.3.1) or tasks that seek to detect security issues and assist with mitigation of on-going attacks (described in § 2.3.2). Though real-

wold telemetry systems may also be leveraged for other purposes (*e.g.,* long-term capacity planning), these two types of tasks have clear high-stakes implications for network administrators and hence have attracted the bulk of research effort in this area.

2.3.1 Performance Monitoring. A large body of work develops systems to detect when performance events such as packet loss or increased latency occur and to identify the particular subsets of traffic (e.g., a particular set of flows) impacted. The results of this monitoring often take the form of alert-producing systems that log performance events for network administrators to use when debugging reported application-level performance issues. For example, if a client is experiencing issues with freezing video conferencing, the network administrator might check the logs of the network performance monitoring system to determine if lost packets of increased forwarding latency were the cause of the freeze or if a software-level issue is at fault. This section focuses in particular on the recent subset of performance monitoring systems that leverage programmable switch hardware for in-network aggregation.

2.3.1.1 Detecting performance events. The first group of performance monitoring systems relies entirely on switch hardware to observe network traffic and to perform initial aggregation computations, typically grouping packets into flows and reporting flow-level performance information to a collector for post-processing and event reporting.

Common design patterns. The systems in this section filter and aggregate particular performance-related events in switch hardware, then use CPU-based software (running on the switch's CPU and/or on a centralized collector) to correct for possible errors made by the lossy hardware approximations (e.g., hash collisions). The network-wide perspective along with inherent redundancy in the targeted data

center networks allows disentangling such errors in way which are not possible in the single-switch scenarios considered in § 2.2.

Key contributions. FlowRadar [100] and LossRadar [101] make the primary contributions in this group leveraging Invertible Bloom filter Lookup Tables [69] and Invertible Bloom filters [56] respectively. These works capture per-fivetuple counters over very short time scales (e.g., 10 ms) using encoded flowsets collected in switch hardware and network-wide decoding. The fine aggregation as well as temporal granularity enables detecting a wide range of performance events (in the CPU-based post processing layer) including transient block holes, forwarding errors, ECMP load imbalance.

Rather than exporting flow-level metrics and performing event detection in post-processing, several more recent works seek to execute event detection logic directly in the data plane and only export records describing particular detected events to a central collector. BurstRadar [85] initiates this trend by detecting microbursts at port-level in the egress pipeline and emitting grouped burst snapshots summarizing all packets responsible for the burst. HyperSight [185] and NetSeer [186] extend this idea to several other performance events (including excessive forwarding delay, degraded throughput, long queues, out-of-order packets, and packet loss) by using a novel Bloom filter queue algorithms and circular-buffer aggregation respectively. PacketScope [158] offers even deeper visibility into performance events on a single switch by extending Sonata's [70] stream processing language with primitives connected to different pipelines and queuing stages within the switch ASIC.

Open problems. A key open problem with works in this category is how to connect performance events detected at the network level to higher-level performance metrics (e.g., at the application level). A somewhat related open problem is that although the

works in this section target data center networks in particular, they do not provide comprehensive support for integrating with and interpreting the complex layered logical structure of data center traffic, for example virtual private clouds as discussed in [128].

The second group of performance monitoring systems observes network traffic both in switch hardware programs as well as in the network stacks of end hosts. This allows the system to augment the packet processing and aggregation capabilities of switch hardware (which have strict limits on memory and number of operations) with more flexible CPU and memory resources on end hosts.

Common design patterns. Works in this group coordinate per-flow, per-switch monitoring by using in-band methods initiated from and collected at end host network stacks. Packets leaving end hosts may be tagged with instructions for what performance metrics are required encoded in custom-designed header fields. In addition to in-network aggregation computations, switches parse and exchange information via these header fields. Ultimately packets returning to end hosts carry the desired performance data up to the CPU-based network stack where further aggregation and post processing can occur. In addition to expanding the processing flexibility and memory capacity via computation at distributed end-host CPUs, this approach also enables integrating network-stack and application-layer information into the performance monitoring results.

Key contributions. The first work in this category OmniMon [80] develops a careful method of splitting per-flow, per-switch packet count monitoring across switches and end hosts with a focus on maintaining per-epoch consistency across different observation points. In particular, they propose a hybrid consistency model (where each packet belongs to the same epoch) as a practical yet useful relaxation of strict

consistency. This enables detailed detection and localization of performance events such as packet loss. LightGuardian [183], on the other hand focuses on reducing communication overheads between switches and end hosts by transmitting aggregate per-flow, per-switch performance metrics collected on switches to hosts in a novel "sketchlet" format. End hosts can incrementally aggregate received sketchlets to construct a detailed view of packet loss, latency, and jitter metrics in the network.

BufScope [64] further leverages end-host participation to attach application-layer metrics (in particular, request ids) to per-flow, per-switch metrics aggregated on switch hardware. They integrate these methods in a comprehensive system that ultimately provides queuing latency measurements for every buffer along the path of an application request.

Open problems. Although works in this section demonstrate that leveraging end-host participation in network performance monitoring has some significant advantages, it also potentially introduces new issues. In particular, the increased system complexity leads to an increased number of potential problems that could impact network performance monitoring such as end host hardware and/or software failures.

2.3.1.2 Diagnosing performance events. In addition to detecting the occurrence of and connections impacted by network performance anomalies, a complementary group of efforts seek to provide a basis for identifying the root-cause of such anomalies. Such root causes typically go beyond the detailed performance metrics of other works by attempting to find larger-scale correlations between performance-related events.

Common design patterns. Works in this sections trigger collection and/or reporting of telemetry data only when a performance event occurs via automatic or manual trigger conditions. Though these systems may compute metrics continuously in switches,

computations and communications specific to debugging the root cause are only executed when debugging is activated. The key challenge is to strike a balance such that a minimal amount of computation overhead is required in the non-debugging state, but a maximal amount of information can be gathered and correlated when the debugging state is entered.

Key contributions. The first work in this area, SwitchPointer [156], builds on previous state-less tracing approaches [154, 155] to combine aggregation in switch hardware with path-based debugging. In particular, SwitchPointer uses switches as a "directory service": when a performance event occurs and the system enters debugging mode, the pointers stored in each switch along the problematic path are used to lookup per-flow counters maintained in relevant end hosts. Combined with a novel hierarchical division of time, this enables quick answers to questions such as which other flows were sharing a link with flow X at time t (e.g., when flow X experienced unusual queuing delay).

SpiderMon [166], on the other hand, automatically detects when flows experience high cumulative queuing latency by summing the time spent by individual packets in each queue along their network paths. When a path experiences abnormally high latency, SpiderMon triggers export of flow-level metrics along all paths intersecting the impacted path by using novel "spider" control packets exchanged between switches. A central controller receives reports generated from switches that process spider packets and (re)constructs debugging results for the impacted path.

Open problems. A key open problem in this group of works is justifying the need for extra complexity to answer questions that are theoretically already covered by simpler performance monitoring systems like FlowRadar [100]. Intuitively as networks increase in scale and complexity (e.g., total number of nodes), the overheads of

always-on systems increase at a much faster rate compared to the on-demand systems considered in this section. However, it's unclear where the advantages tip from always-on to on-demand and what other aspects of the network setting may impact this trade-off (e.g., presence of virtual routing [128], software gateways [167]).

2.3.1.3 Takeaway. Although the efforts described in this section provide effective performance monitoring solutions for certain types of data center networks, it is unclear how their contributions can be applied to more complex types of traffic observed on edge, enterprise, and ISP networks (*i.e.,* connection between **Area 1** and **Area 3**). In particular, common techniques proposed in these works, like leveraging end-host CPUs and distributing query processing over a large number of on-path switches, may not be feasible in these other network settings. Even in the data center context, it is unclear how these proposals relate to the known multi-layered structure of real-world data center traffic (*e.g.,* with virtual routing, heterogeneous network processing devices, large numbers of application-layer protocols).

2.3.2 Security. Despite supporting a wide range of mission-critical services, modern networks are not built with security as a primary design concern. As a result, organizations must carefully monitor traffic flowing through their networks to identify potentially malicious activities. Given the promise of highly efficient traffic monitoring, a large body of work seeks to leverage programmable switch hardware as the primary processor for network security event detection systems. In § 2.3.2.1 we first consider how several security event detection tasks have been expressed and implemented using stream-processing telemetry systems (introduced in § 2.2.2), then consider the particular task of mitigating volumetric DDoS attacks in § 2.3.2.2.

2.3.2.1 Detecting security events. Particular forms of malicious traffic, such as network-based attacks, can be understood as producing distinct traffic

signatures that can be detected by carefully crafted queries. A particular collection of such queries developed as part of the Sonata project [70] and publicly released [16] has become an unofficial standard bench mark for telemetry systems that are capable of computing this particular type of stream-processing query [80, 116, 187]. As shown in Table 1, these queries are based on detecting particular types of suspicious behavior by using different packet filterings, groupings, aggregation metrics, and thresholds.

Query Name	**Description**	**# in [16]**
DDoS	Find dests. that recv. from large number of sources.	5
Port Scan (PS)	Find sources that send to large number of ports.	4
Super Spreader (SS)	Find sources that send to large number of dests.	3
TCP New Cons. (TNC)	Find dests. that recv. large number of TCP SYN packets.	1

Table 1. Examples from the "Sonata" [70] security event detection queries.

Key contributions. The key advantage of the Sonata queries is that they provide several concrete examples of how high-level security event detection goals can be translated into low-level traffic query specifications. This enables direct performance comparisons between different approaches to implement query processors in terms of accuracy and overheads.

Open problems. The Sonata queries only provide a single, highly-specific traffic query example for each of the security event detection tasks considered. Because of this, they lack in-depth consideration of the particular high-level security events evoked and in many cases may also detect traffic patterns that are not actually associated with the chosen security event (*i.e.,* high false-positive rates). For example, the DDoS query simply detects servers that receive traffic from a large number of clients, which in certain network settings could also be observed under normal, non-suspicious

circumstances like flash-crowds. Because of this, the real-world utility of the Sonata queries is uncertain.

2.3.2.2 Volumetric DDoS Defense.

Volumetric distributed denial of service (DDoS) attacks sever or degrade network connections by flooding particular network links or end hosts with large volumes of traffic sent from a large number of distributed sources [160, 138, 137, 168, 165, 164]. A variety of closed-loop systems have been proposed to defend against volumetric DDoS attacks by monitoring traffic to detect particular subsets of traffic associated with DDoS techniques and actively rate-limiting or blocking this traffic directly in programmable switch hardware. Volumetric DDoS related tasks on programmable switch hardware have seen disproportional research effort compared to other security tasks due to the inherently large traffic volumes associated with volumetric DDoS attacks and the high packet-processing efficiency of programmable switch hardware.

Fine-Grained, CPU-Controlled Languages. The first group of works seeking to develop in-network DDoS defense using programmable switch hardware leverages hardware to aggregate and detect attack traffic, but also relies on a logically centralized controller (running in a CPU-based system) to coordinate deployment of attack mitigation operations in switch hardware.

Common design patterns. The works in this section propose policy languages or APIs with several primitive operations (e.g., SYN cookies, block-lists) which network administrators can use to construct DDoS mitigation mechanisms tailored to particular attack behaviors. The switch hardware implementations of these mechanisms typically use similar sketch-based approximate algorithms to deal with constrained resources.

Key contributions. Poseidon [181] introduced the first policy-language for switch hardware-based DDoS defense which includes flexible predicates to identify common attack packets (e.g., SYN packets, UDP response packets), counters to group and aggregate matching packets, and a set of threshold-based actions including dropping and rate-limiting detected attack traffic. Similar to Sonata [70], defense policies are implemented by partitioning primitive operations across programmable switch hardware and CPU-based servers. Jaqen [103] extends the approach put forth in Poseidon by adding resource-efficient approximate switch hardware primitives tailored for particular attack vectors such as "UnmatchAndAction" for reflection-based attacks and two custom SYN-proxy designs. Ripple [174] also implements a stream-processing style mitigation policy language on programmable switch hardware, but leverages the concept of a global panoramic view of traffic (maintained by a distributed synchronization protocol between switches) to realize distributed defense against a particular class of dynamic link-flooding attacks (in particular attacks like Coremelt [153] and Crossfire [86]).

Open problems. Although Poseidon [181] and Jaqen [103] both discuss the need to dynamically reconfigure attack mitigation in the face of modern dynamic DDoS attacks, neither provides adequate mechanisms for doing so. Jaqen does include attack detection through a refined version of UnivMon [102], but it's unclear how this detection capability pairs with the proposed mitigation mechanisms. Both methods require compiling and installing defense mechanisms in switch hardware which induces several seconds of downtime during which no traffic can be processed on the switch requiring re-routing during downtime.

These methods also propose developing finely-tuned mitigation mechanisms for specific attack types, leaving open the more fundamental issue of how to defeat the

inherent imbalance the underlies DDoS's effectiveness—it's very easy for the attacker to come up with a new attack vector (or modify a known vector) whereas it's very hard for a network administrator using one of the systems discussed in this section to anticipate and deploy mitigation methods for all possible attack vectors.

Coarser-Grained, Switch-Only Policies. The second group of works seeking to develop in-network DDoS defense implements all stages of attack traffic aggregation, detection, and mitigation entirely in the switch hardware pipeline. The key advantage of these approaches is the ability to quickly react to attack occurrences and changes in attack traffic without control plane involvement and/or needing to re-compile and reload new mitigation mechanisms.

Common design patterns. These works propose comparatively generic methods to both detect attack occurrences and to separate attack and benign traffic (that can still be implemented entirely in switch hardware). Due to the possibility of false positives, they propose rate-limiting the detected attack traffic rather than outright blocking or allowing as considered in the previous subsection.

Key contributions. Euclid [52] describes the first DDoS detection and mitigation method that operates entirely in a single switch hardware program. The proposed method uses sketches to estimate source and destination address entropy (following a similar design as [94]). Relative changes in entropy trigger attack detection after which Euclid applies rate limiting to the set of source, destination pairs identified as being most responsible for the change in entropy. ACCTurbo [24] describes an alternative approach to attack mitigation entirely within switch hardware by adapting the idea of aggregate-based congestion control [107]. In particular, ACCTurbo performs simple clustering in switch hardware and leverages a CPU-based control plane to install rate-limiting rules on large aggregates that may be associated with attack traffic.

Open problems. Although the works considered in this section effectively develop DDoS mitigation approaches that can function at runtime without interrupting packet forwarding, they do not offer the same level of precision as Poseidon [181] or Jaqen [103] or a path towards distributed mitigation as in Ripple [174]. A key open question then is how to improve the precision of these methods while still working in the confines of switch hardware.

Additionally, the design pattern of maintaining all attack mitigation related state entirely in the data plane has a side effect of preventing network administrators from observing the mitigation system's actions (e.g., to ensure mission-critical traffic is not accidentally blocked by mitigation). Since Euclid acts on a packet-by-packet level and does not store particular source and destination addresses, it is literally impossible to assess what traffic it will rate-limit or block. ACCTurbo improves on this situation somewhat by exposing aggregate-level information to the control plane, but does not offer motivation for why particular aggregates should be considered attack traffic—in particular, the aggregates reported by ACCTurbo may contain a mix of both attack and benign traffic since switch resource limitations require relatively coarse-grained aggregates compared to the source, destination level considered in Euclid. Overall, these observations motivate an additional open problem of how to balance interpretability of decisions made by these types of defense systems with sufficient precision of mitigation actions.

2.3.2.3 Takeaway. Although the efforts described in this section provide novel applications of telemetry system capabilities to address important security problems, their practical discriminative value and robustness in the face of real-world traffic are unclear (*e.g.,* connection between **Area 1** and **Area 3**). In particular, tuning thresholds (*e.g.,* for the security event detection queries proposed in

Sonata [16]) induces non-trivial effort for network administrators and may ultimately still not be precise enough to detect important security events. Moreover, proposals to leverage sketches-based approximate defense methods directly in switch hardware lead to highly unpredictable effects on the traffic of benign users.

This section examined state-of-the-art research efforts in each of the three areas outlines in Chapter I and described how limitations or open problems in each area can be understood as "missing links" between areas. In the next three chapters, we will outline several concrete contributions made to state-of-art by more deeply leveraging these connections.

CHAPTER III

FOUNDATIONS: CHARACTERIZING TRAFFIC STRUCTURE

The content around Figure 6 and Figure 7 are from [116]. Application of the method of moments for multifractal analysis of IP addresses discussed in § 3.2 was originally developed by Walter Willinger and Arpit Gupta. The text in § 3.2.1 was initial composed by Walter Willinger and adapted for this context. Walt O'Connor helped develop the implementation used to produce Figure 17 and Figure 18. I wrote all other analysis code, figures, and text in § 3.2.2. Ramakrishnan Durairajan, Reza Rejaie, Arpit Gupta, and Walter Willinger assisted with editing.

This chapter presents statistical characterizations of two different aspects of network traffic that are critical to telemetry system design and implementation. In particular, § 3.1 studies the time series formed by the number of distinct groups per time unit and § 3.2 studies the distribution of observed IP addresses in the address space. As discussed in § 1.4, the outcomes of both studies shape design strategies pursued in Chapter IV.

3.1 Temporal Structure

As discussed in § 2.1.1, the time-domain structure of the number of distinct traffic groups observed per time unit has received little to no attention in prior research efforts despite its direct relevance to telemetry system design. To study this structure, we examine a simple random sample of 28 days drawn from MAWILab's [61] 2015 dataset. Each day consists of a 15 min trace starting at 2 pm of all packets observed on a trans-Pacific link between the US and Japan. To preserve user privacy, packet payloads are not included and all addresses are anonymized.

We consider three different groupings of packets based on the example Sonata queries shown in Table 1, in particular *(i)* the number of distinct source address,

destination port pairs for the "port scan" (PS) query, *(ii)* the number of distinct source, destination address pairs for the "DDoS" and "super-spreader" (SS) queries, and *(iii)* the number of distinct destination addresses that receive at least one TCP SYN packet for the "tcp new connections" (TNC) query. For each grouping, we count the number of distinct groups that are observed in contiguous 5-second time windows or *epochs*.

To illustrate a case with extreme changes in the number of groups, we consider a member of our sample from Nov. 14th. As shown in Figure 6, this excerpt starts with relatively stable traffic, then suddenly, due to an actual DDoS attack or other causes (which we do not claim to identify), around the 20^{th} time window contains an *order of magnitude* large number of sources sending regular pulses of traffic.

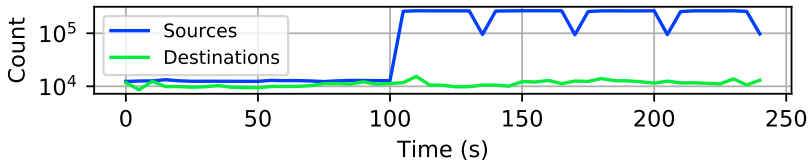
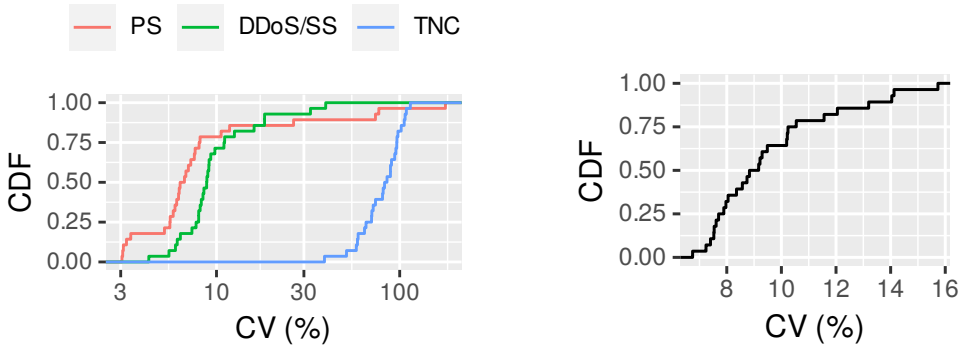


Figure 6. Number of distinct sources and destinations in excerpt from MAWILab data on Nov. 14th, 2015.

To summarize the extent of temporal dynamics in the number of groups over our entire sample of 28 traces, we compute the CV of number of groups per-epoch and number of packets per-epoch for each trace. Figure 7 shows the distribution of CVs over all traces in our sample for group-based counts (Figure 7a) and packet counts (Figure 7b). The CVs of the number of groups for PS and DDoS/SS cluster around ∼10%, however, group-based CVs have a much longer tail including traces with CV over 100%. (For reference, the CV of the single trace of sources per time unit shown in Figure 6 is ∼52%.) In the case of TNC, the group-based CV is consistently high (median of ∼84%), likely a product of how this query only looks at SYN packets and

each SYN packet is typically associated with a new flow. These results illustrate that contrary to the assumptions of prior work outlined in § 2.1, in real network traffic there is significant variance in the number of distinct groups observed each epoch.



(a) Number of distinct keys per epoch.

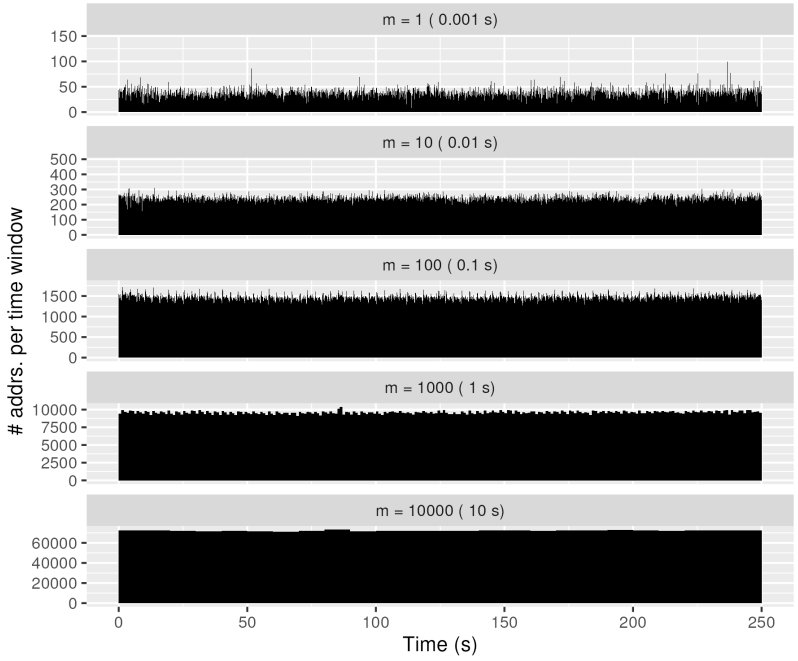
(b) Number of packets per epoch.

Figure 7. Distribution of CVs over our sample of 28 traces from MAWILab [61].

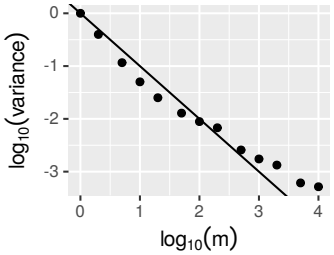
Given the well-establish self-similar structure of traffic rates (*e.g.*, packets per time unit), a logical next step is to investigate whether or not the number of distinct groups per time unit might also exhibit self-similar structure. Following common methods [97], we define the aggregate time series $X'^{(m)}(t)$ to be the number of groups in non-overlapping blocks of size m at time t. Then, $X'^{(m)}$ is asymptotically self-similar if the correlation structure of $X'^{(m)}$ approaches the correlation structure of $X'^{(1)} = X'$ as $m \to \infty$. Intuitively, this implies that $X'^{(m)}$ will look qualitatively similar for different values of m.

To illustrate, we focus on a different trace in our sample from MAWILab captured on March 3rd (which has more typical traffic compared to Nov. 14th) and let $X'^{(m)}$ be the number of distinct source addresses per time unit with the smallest time division $m = 1$ corresponding to 1 ms. Figure 8a shows $X'^{(m)}$ for a few selected values of m over the first 250 s of the trace. We observe that the relatively rough,

bursty appearance of the finest time granularity $m = 1$ gradually flattens out under increasing time aggregation as we increase m. To confirm, in Figure 8b we draw a variance-time plot [97] of $X'^{(m)}$ for a wide range of values of m and show that the slope of the normalized variance is close to -1 on a log-log plot which indicates statistically evidence for $X'^{(m)}$ *not* being self-similar.



(a) $X'^{(m)}$ for several m.



(b) Variance-time plot showing slope near -1.

Figure 8. Initial analysis of the number of distinct source addresses per time unit. The qualitative "flattening" of the timeseries for larger m indicates lack of self-similar structure.

We applied the same analysis of Figure 8 to all traces in our sample from MAWILab and confirmed that in typical cases where there are no large, sudden changes in the number of observed addresses, X' is not self-similar. Cases with large changes, such as the Nov. 14th example shown in Figure 6, are harder to analyze because the observed changes are highly localized to particular time windows which likely correspond to particular non-typical events (*e.g.,* network-based attacks). As a

result, variance-time plots do not show consistent asymptotic behavior as m increases, but expose different scaling regimes at different intermediate values of m based on position and duration of the changes within the 15-minute trace excerpt.

The key implication of this initial analysis is that although typical-case behavior of the number of groups per time unit (X') does not exhibit the same complex self-similar structure of data rates like packet count per time unit (X), pervasiveness of atypical events in real network traffic implies that the number of groups per time unit still has significantly high variance. Further analytic work is required to establish a more satisfactory statistical characterization of the time-domain structure of the number of groups per time unit. In particular, such analysis will likely need to revisit existing physical models of network time series (*e.g.,* the Poisson cluster model proposed in [75]) with an eye towards more realistic distributions of address-arrival and address-departure processes instead of the current focus on flow-arrival processes. Moreover, longer-duration traces (lasting days or weeks) will also be critical for understanding the dynamics faced by telemetry systems deployed in real-world networks which would be expected to remain accurate and efficient for comparable durations. This dissertation leaves such analysis for future work and focuses instead on an initial goal of leveraging the observation of time-domain structure from Figure 7a—that the number of groups has high variance (regardless of its scaling behavior). Even if these time series do "flatten out" at some particular longer duration of aggregation, telemetry systems will still have to cope with these persistent and extreme short-term changes in number of distinct groups. In § 4.1 we will develop several key techniques to handle these kinds of short-term dynamics in telemetry system design through a generic and non-parametric method that does not require a priori analysis of their temporal structure.

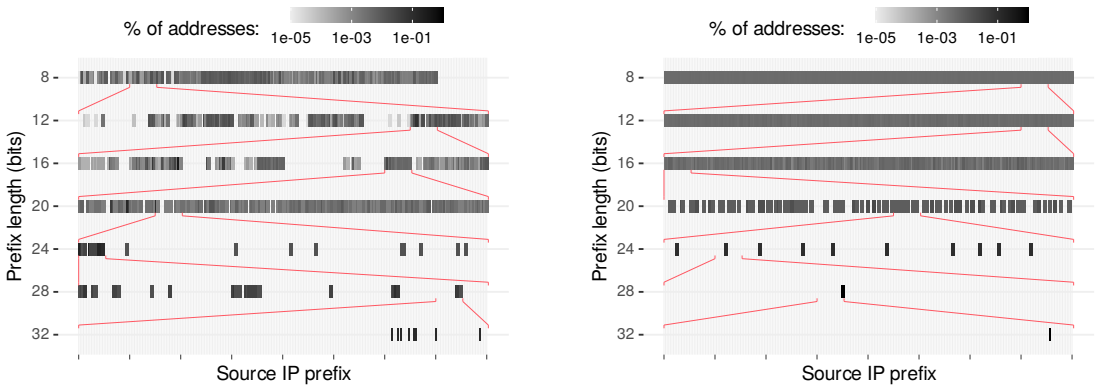
3.2 Spatial Structure

This section develops an initial characterization of the spatial structure of observed IP addresses, which, as discussed in § 2.1.2, has received little research attention since initial results some 20 years ago. We first provide the necessary background for understanding the multifractal structure of IP addresses in § 3.2.1. Next, we investigate data sets associated with the IP address allocation process to show how IP allocation is consistent with conservative cascade statistical models in § 3.2.2. Finally, we present empirical evidence for multifractal scaling in a range of real-world data sets in § 3.2.3.

3.2.1 An Echo from the Past. The preliminary findings about the multifractal structure of the observed IP addresses in measured network traffic originally reported in [90, 91, 31] rely on traffic traces that have been collected some 20-25 years ago. To address the dated nature of these traces, we first set out to assemble our own repository of more recently collected Internet traffic traces. A detailed description of this repository can be found in Appendix A, with Table A.1 listing all the traces and associated metadata.

3.2.1.1 A picture is worth a thousand words. For effective 1D visualizations involving IPv4 addresses, it is common to identify the entire IPv4 space with the unit interval $[0, 1)$ using the standard CIDR notation. Per this notation, the unit interval $[0, 1)$ represents the entire IPv4 space, or /0, and there is a 1-1 correspondence between the l-th dyadic partition P_l of $[0, 1)$ (i.e., the collection of intervals $E_{l,i} = [i2^{-l}, (i + 1)2^{-l}); i = 0, 1, \ldots, 2^l - 1$ and the set of all 2^l prefixes of length l $(0 \leq l \leq 32)$. A similar identification of the IPv6 address space is likewise constructed with $0 \leq l \leq 128$.

With this mapping between the IPv4 space and the unit interval in 1D, we consider for illustrative purposes one of our traces, CAIDA-dir-A, listed in Table A.1 in Appendix A and form the dataset CAIDA-dir-A100 that consists of the first $100,000$ unique source IP addresses observed in CAIDA-dir-A. We also generate a same-sized synthetic dataset UNIFORM100 by casting 100K 32-bit integers as IP addresses and drawing them from the uniform distribution on the unit interval. Next, building on the intuitive notion that multifractal structure manifests itself in a pronounced "cluster-within-cluster" or "nesting of structure" property, we realize this intuition pictorially with a series of "zoom-in" steps that reveal how clusters of observed IP addresses at a given scale (e.g., /16 subnet granularity) appear when viewed under a microscope; that is, when examined at finer scales (e.g., /17 or finer subnet granularity). Figure 9 illustrates seven steps of this "zoom-in" process for the measure μ_A that counts the number of observed IP addresses in each subset (e.g., prefix) of the set CAIDA-dir-100A derived from for the real-world traffic trace CAIDA-dir-A (Figure 9a) and the synthetic set UNIFORM100 (Figure 9b), respectively.



(a) CAIDA-Dir-A100k. (b) UNIFORM100k.

Figure 9. 1D Pictorial evidence of multifractal vs not-multifracatal IP address structure, using the dataset CAIDA-Dir-A100 (left) and UNIFORM100 (right), respectively.

In the case of the CAIDA-dir-A100 dataset (Figure 9a), with the possible exception of the top row where artifacts due to real-world IP allocation (e.g., unassigned address space) are visible and the bottom row where the discrete nature of the IPv4 space imposes a strict limit to the described fragmentation process (i.e., the largest possible prefix length is 32 which defines individual IP addresses), the zoom-in's for the intermediate stages look qualitatively similar, exhibit visually apparent cluster-within-cluster behavior that defies the identification of a "typical" cluster size, and provides pictorial evidence that the observed IP addresses are highly intermittently or irregularly distributed across the entire IPv4 space, consistent with a multifractal address structure. When compared with the UNIFORM100 dataset (Figure 9b), we also observe qualitatively similar-looking zoom-in's across the different rows, but the appearance of the zoom-in's is noticeably "less irregular" – the cluster-within-cluster behavior is more regular, allows for discerning a "typical" cluster size for each row, and results in the observed IP addresses being evenly distributed across the IPv4 space.

3.2.1.2 From "visual" to statistical evidence. The "pictorial proof" in Figure 9 suggests that the observed IP addresses in measured Internet traffic are distributed across the IPv4 space in a highly irregular or intermittent manner, symptomatic for an address structure that exhibits the hallmarks of multifractals. These visuals could therefore be viewed as a strong argument for pursuing as obvious next step an in-depth statistical analysis of the listed traffic traces so as to determine more objectively whether or not the shown visual evidence is indeed consistent with a multifractal address structure. In theory, such an analysis would allow us to state more confidently whether or not the obtained visual evidence can be assumed to have resulted from an underlying mathematical model that supposedly generated

the observations behind the shown visual evidence in the first place and can be theoretically shown to be multifractal.

However, rather than pursuing this obvious next step and performing an in-depth statistical analysis of our collection of recent traffic traces, we decide to instead embark on an in-depth physical explanation that attempts at getting at the root cause of why observed IP addresses in measured Internet traffic might exhibit multifractal scaling behavior. This strategy has numerous advantages over the more conventional approaches that emphasize utilizing state-of-the-art statistical theory and the latest statistical toolboxes to perform an in-depth statistical analysis of the new traffic traces and obtain "hard" statistical evidence whether or not the traces are consistent with multifractal IP address structure. For one, the development of a statistical theory for a multifractal analysis of real-world data is still in its infancy, with only few known results for quantifying the confidence in computed estimates (e.g., determining confidence intervals) or computing the power of hypothesis tests (e.g., assessing a chosen model's goodness-of-fit). As a result, in practice, multifractal analysis of real-world data relies on empirical techniques and toolboxes, which leave the obtained findings open to interpretation, makes them sensitive to known or unknown idiosyncrasies in the data, and produce in general inconclusive results. On the other hand, a mathematically rigorous and empirically validated physical explanation of observed multifractal structure in real-world data that can withstand scrutiny by domain experts generally eclipses statistical arguments and is typically viewed as providing a conclusive if not the final answer.

Mathematical interlude: multifractal analysis in a nutshell. Investigations of the spatial aspects of measured network traffic are typically concerned with entities such as (finite) measures μ defined on a subset E of the real line. Following [140],

it is common to study the irregularities or singularities of such measures at a point $x \in E$ mathematically in terms of a pointwise scaling exponent called the **Hoelder exponent** $\alpha(x)$ satisfying $\mu(B(r,x)) \approx r^{\alpha(x)}$ for sufficiently small r-values, where $B(r,x)$ denotes a ball of radius $r > 0$ around $x \in E$. The measure μ is called **multifractal** if this Hoelder exponent varies as x varies; μ is **monofractal** if a single Hoelder exponent α_0 describes the singularities of μ [126]. **multifractal analysis** is concerned with effectively summarizing the complex and seemingly erratic nature in which the Hoelder exponent can fluctuate for different values of $x \in E$ and providing practitioners with a succinct global description of these fluctuations.

One such description is geometrical in nature and is based on the so-called **histogram method** that uses the concept of **fractal dimension**, computes the **multifractal spectrum** $f(\alpha)$ as the fractal dimension of the iso-scaling set $E_\alpha = \{x \in E, x \text{ has Hoelder exponent } \alpha\}$, and can thus be understood as providing a parsimonious quantitative digest and microscopic characterization of the complex scaling behavior exhibited by the measure μ. Unfortunately, accurately inferring the multifractal spectrum by reliably estimating the Hoelder exponent point by point from real-world data and then computing the fractal dimension of each resulting iso-scaling set E_α is fraught with problems to the point of being impractical.

An alternative description that is statistical in nature and also practical, known as the **method of moments**, is concerned with computing the so-called **structure function** $\tau(q)$ that describes the scaling behavior of the higher-order sample moments of μ. This method can be understood as providing a compact but macroscopic characterization of the scaling behavior of the measure μ [140].

The two descriptions are related thanks to the so-called **multifractal formalism** [126, 135] that asserts that under certain technical conditions, the

multifractal spectrum $f(\alpha)$ is the **Legendre transform** of the structure function $\tau(q)$ (and vice versa); that is, $f(\alpha) = \inf_q(q\alpha - \tau(q))$. This relationship is especially useful in practice when all that is needed from a multifractal analysis of real-world data is plausible evidence whether or not the data is consistent with multifractal scaling behavior. Indeed, by virtue of being the Legendre transforms of one another, a non-linear structure function $\tau(q)$ implies a non-degenerate multifractal spectrum $f(\alpha)$ that is defined for some continuous range of α-values on the real line. Thus, to obtain plausible evidence of multifractal vs monofractal, it suffices to check whether the structure function $\tau(q)$ is not linear (evidence for multifractal scaling) or linear (evidence for monofractal scaling).

3.2.1.3 Conservative or semi-random cascades. Mathematical modeling of observed multifractal scaling behavior in various real-world datasets was pioneered by B. Mandelbrot who first introduced a class of prescriptive or evocative models known as **multiplicative cascades** [109]. Following [134], a multiplicative cascade is an iterative process that fragments a given set into smaller and smaller pieces according to some rule and, at the same time, distributes the total mass of the given set according to another rule. The cascade's **generator** determines the redistribution of the set's total mass at every iteration and can be either deterministic (as, for example, in the "Cantor dust" model considered in [90]) or random as in the case of the multiplicative, multiscale innovation model (e.g., see [124, 67, 31]). Multiplicative cascades with the property that the generator preserves the total mass of the initial set at each stage of the construction are called **conservative cascades** or **semi-random cascades**, are of particular interest for capturing the complex scaling behavior observed in actual measurements of many naturally occurring phenomena. Under certain technical conditions, they can be shown mathematically to result in

limiting objects that represent well-defined multifractals that can be characterized in terms of the cascade's generator, which in turn can be inferred for real-world data (e.g., see [134] and references therein).

Though conservative cascades have been extensively explored [58, 98, 110, 136] and debunked [162] in the analysis of the *temporal* nature of Internet traffic, their potential to model *spatial* aspects of addresses observed in Internet traffic, which is the focus of our work, has received less attention.

IP address allocation as a conservative cascade process. The impetus of our efforts comes from a speculation in [90] that the fact that multifractals can be constructed by means of certain conservative cascade processes brings to mind the way that IPv4 addresses are typically allocated in the Internet: ICANN assigns big IP prefixes to the Regional Internet Registries (RIRs), the RIRs allocate blocks to ISPs, who further allocate sub-prefixes to their customers, and so forth. While the way of allocating IP addresses may differ for each of these stakeholders and may reflect historical, social, or economics-based aspects, it is plausible to assume that the root cause of the observed multifractal scaling behavior in the sets of encountered IP addresses in measured Internet traffic is indeed an underlying conservative cascade process that captures the essential features of IPv4 address allocation in the modern Internet.

3.2.2 Physical Explanation and Validation.

In this section, we investigate the root cause of observed multifractal IP address structure and argue that the processes behind allocation are fundamentally aligned with the conservative cascade model, making it an ideal tool for analysis of observed address structure. We follow the classic "three-level" story of how IP addresses are allocated in the modern Internet shown in Figure 10: IANA makes "global" allocations to RIRs (§ 3.2.2.1),

RIRs make "first-level" allocations to particular organizations (§ 3.2.2.2), and large organizations (ISPs, CSPs, etc.) make allocations to their customers (§ 3.2.2.3). At each level, we illustrate how the conservative cascade model can be viewed as a direct result of the allocation policies made by the particular entities in control of that level.

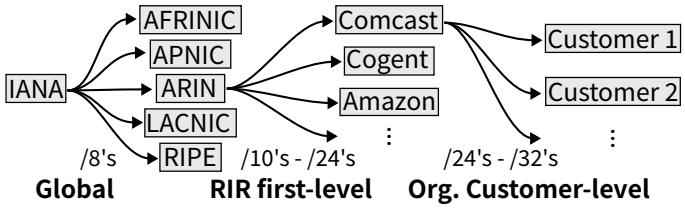


Figure 10. High-level "cascade" process of Internet address allocations.

3.2.2.1 ICANN/IANA and the RIRs: IPv4 space at coarse granularity. The top-level authority on IPv4 addresses is the Internet Assigned Numbers Authority (IANA) which is an affiliate of the Internet Corporation for Assigned Names and Numbers (ICANN). For the IPv4 address space, IANA publishes an authoritative list of how each possible top-level /8 prefix is allocated [10] known as the "IPv4 Address Space Registry". After allocation of the last free /8 block in 2011 [81], the registry now covers the entire IPv4 address space.

Clustering of blocks allocated to particular RIRs. The fact that all IANA assignments are made at /8 granularity may appear to imply a lack of any structure above this in the prefix tree. However, closer inspection of how /8's are allocated reveals a significant degree of clustering along boundaries implied by shorter prefix lengths (*e.g.,* the /1, /2, and /3 clusters leftover from class-*full* network allocation or induced by reserved spaces).

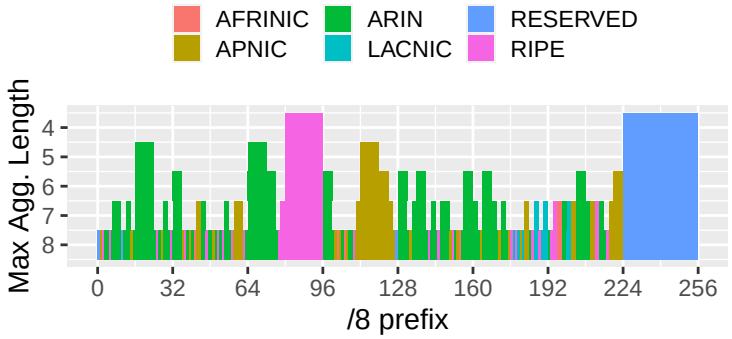


Figure 11. Global view of IANA IPv4 Address Space Registry showing which regions of the space are allocated to several primary RIRs and the maximum possible aggregation of each contiguous per-RIR block.

Figure 11 shows which RIR is associated with each /8 block allocated by IANA[1] (x-axis) and how much aggregation is possible in each contiguous cluster of blocks allocated to the same RIR (y-axis). Note that smaller Max Aggregation Lengths indicate shorter prefixes with more aggregation. The largest possible aggregation (outside of reserved space) is RIPE's cluster around 80.0.0.0/4 and both ARIN and APNIC also have large /5 clusters. Our analysis also shows that type of allocation has clear clustering along the historic class-full network boundaries and that several clusters were allocated sequentially over several years indicating that IANA likely pre-allocated regions for particular RIRs (at shorter than /8 granularity).

3.2.2.2 Allocation policies of ARIN: IPv4 space at medium granularity. We next investigate how the allocation policies of a particular RIR, in this case ARIN, can be understood in terms of the conservative cascade model. Allocation decisions made by RIRs are kept track of in the WHOIS database which records which organizations are responsible for which network address ranges. We

[1]We use the WHOIS field of each block instead of the Designation field here since several /8 prefixes are still listed as allocated to individual companies or organizations (*e.g.,* the US Department of Defense) while delegating WHOIS record management to a parent RIR.

collect the bulk WHOIS dataset from ARIN [3] which currently contains ∼3.2M IPv4 networks with network sizes ranging from /8 to /32 with median at /29.

WHOIS records registered with ARIN can be broadly grouped into two categories: "first-level" assignments which represent allocations of address ranges to particular organizations and "organization-level" assignments which are registered on behalf of ARIN's clients and represent internal divisions in how these organizations use their assigned regions (*e.g.,* different address prefixes often correspond to different geographic regions) or customers of these organizations. Here, we focus on first-level assignments to understand how ARIN divides its assigned range to client organizations, then in § 3.2.2.3 we leverage organization-level assignments to understand how client organizations divide up their assigned address ranges.

Clustering of first-level assignments. First-level records account for ∼2.6% of ARIN's WHOIS records and are primarily direct allocations to a variety of different types of organizations including public universities, medium to large scale corporations, communications companies, and service providers. They tend to have shorter prefix lengths (median 23) compared to records with two or more parents. However, similar to IANA allocations they tend to cluster in implicit groups that imply larger-scale structure down to /10 level granularity. Such implicit clustering of the published first-level allocations reflects how ARIN internally manages its assigned address space as a prefix tree representing an addition step in the cascade process.

To illustrate, we visualize a /7 region of address space managed by ARIN in Figure 12. In this visualization, we draw a rectangle for each first-level network record with the width of the rectangle corresponding to the range of addresses covered on the x-axis, the vertical position corresponding to its prefix length, and the color corresponding to the year of registration. The resulting figure shows not only that

ARIN has different policies in different regions but also that these regions fall precisely on prefix boundaries (shown as vertical grid lines) indicating prefix-level clustering. For example, allocations in 68.0.0.0/10 are large blocks given to large companies like AT&T, Comcast, T-Mobile, and Microsoft where as allocations in 69.0.0.0/11 are smaller blocks given to regional ISPs and smaller non-communications companies.

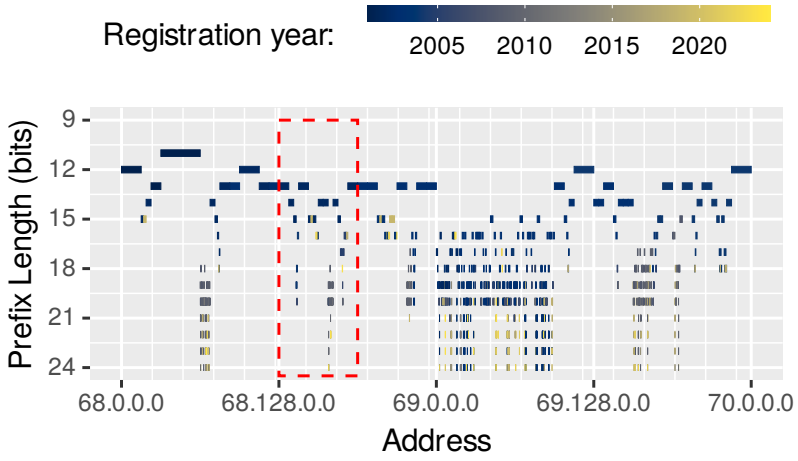


Figure 12. Pattern of first-level network registrations in a region managed by ARIN. (X-axis ticks show /10 blocks.)

Figure 13 further illustrates this prefix-level clustering by drawing the common ancestor prefix tree for 68.128.0.0/10 (outlined in Figure 12). Note that the position of rectangles in this figure are only related to the ordering of addresses while rectangle width is determined by the number of descendants. The strongly unbalanced shape of the tree drawn in this figure illustrates how mass is unevenly distributed in ARIN's allocations with certain intermediate prefixes receiving disproportionately more mass compared to others—a pattern directly consistent with the conservative cascade model.

The WHOIS record tree. A key feature of WHOIS network records is optional inclusion of a "parent" pointer which explicitly identifies another WHOIS network record as being higher in the prefix tree. We use parent pointers to organize the

ARIN WHOIS records into a tree that explicitly describes a cascade process among registered networks. Though our analysis suggests that this tree is wide (with the majority of parents having two or more children) and shallow (with the majority of nodes registered at depth 2), it nonetheless directly confirms the presence of a non-trivial hierarchical cascade in WHOIS records.

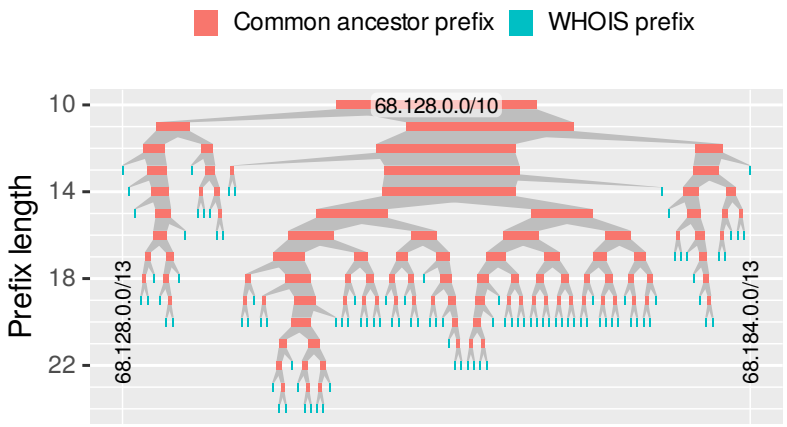


Figure 13. Visualization of common ancestor prefixes for a subset of the region shown in Figure 12.

Temporal stability of ARIN's WHOIS record structure. Given the scarcity of IPv4 addresses in the modern Internet transfer of regions between organizations is a common-place phenomena that could potentially change the prefix-level structure over time. Our analysis reveals that though there are ∼3.5k transfers per year after 2017, the overall impact of these transfers on the prefix-level structure of ARIN WHOIS records is minimal at less than 0.05% annual change in the set of allocated prefixes.

3.2.2.3 Allocation policies of Internet-scale organizations. Beyond the first-level allocations considered in the previous section, we next investigate the structure of allocation policies within individual organizations. We focus on a sample of six large prefixes from different types of organizations since their policies have a

large impact on the overall structure of the address space. In particular, we selected prefixes based on manual inspection of prefixes of 10 bits and shorter, excluding records associated with /8-level RIR structure and ordering by total number of children directly below the prefix. For each selected prefixes we extract all their descendant prefixes from the ARIN bulk whois dataset.

Table 2 lists the six prefixes chosen along with the organization they are (currently) assigned to and the total number of descendant networks. Different types of organizations have different overall numbers of descendant networks; from the telco / ISP companies with $O(10\text{K})$ networks down to the CSP / CDN networks with $O(100)$ networks. Based on "parent pointer" fields in these organization-level WHOIS records, we reconstruct the full record tree for each organization and observe these trees are generally wide (large number of descendants per ancestor node) and shallow (median depth of 3).

Company Name	**Prefix**	**# nets**
Comcast Cable Comm., LLC.	50.128.0.0/9	21127
Verizon Business	63.64.0.0/10	28011
Cogent Comm.	38.0.0.0/8	2195
Level 3	4.0.0.0/9	1478
Amazon Tech., Inc.	3.0.0.0/8	356
Akamai Tech., Inc.	104.64.0.0/10	369

Table 2. Summary of prefixes selected for finer-grained analysis in this subsection.

Despite this "wide and shallow" characterization of the WHOIS records, we observe each organization also has a level of prefix-nesting structure implicit in the placement of their descendant records. We discuss results for the Comcast prefix 50.128.0.0/9 from our sample, but observed qualitatively similar results for other companies investigated.

Figure 14 shows the placement of all descendant records of 50.128.0.0/9 using the same visualization technique as Figure 12. This region of Comcast's address

space shows strong clustering of small /15 through /30 prefixes within several distinct larger prefixes. Moreover these larger-scale clusters appear to correspond to different bursts of allocations though out the period where Comcast controls this space clearly demonstrating that Comcast's internal policy involves dividing their space into prefix-based regions and distributing "mass" (in this case their customer networks) over this larger-scale structure. We also note several empty blocks, *e.g.,* 50.128.0.0/11 and 50.160.0.0/12, which maybe used for internal addresses and hence for which Comcast does not publish public WHOIS records.

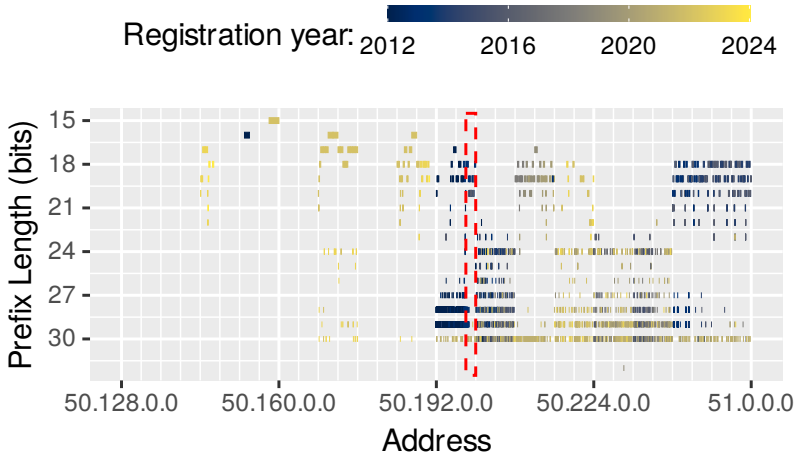


Figure 14. Pattern of network allocations in a /9 prefix registered to Comcast. (X-axis ticks show /11 blocks.)

To further illustrate Comcast's internal prefix-level allocation policy, we explicitly draw the common ancestor prefix tree of 50.198.0.0/15 (outlined in Figure 14) using the same visualization technique as Figure 13 Note that unlike the first-level tree shown in Figure 13, in Figure 15 several parents in the common ancestor tree are also registered in the WHOIS records providing explicit confirmation of our inferences into Comcast's internal policies. For example, the two /18 records with explicit prefix labels in Figure 15—50.198.0.0/18 and 50.198.64.0/18—are named CBC-ILLINOIS-14 and CBC-NEW-ENGLAND-24 respectively and their children are named for

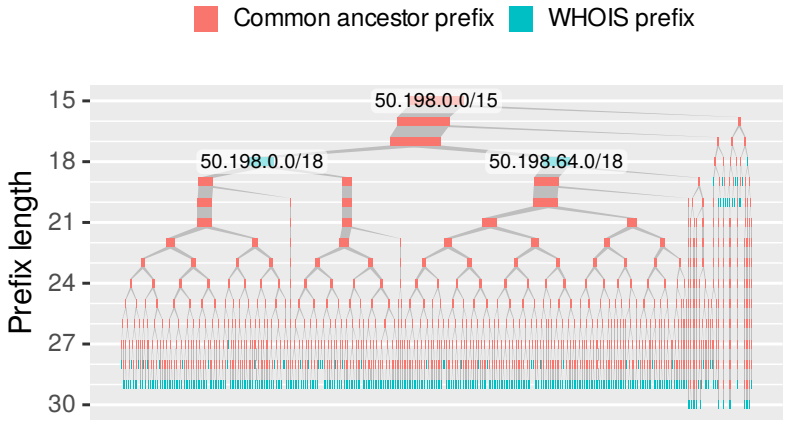


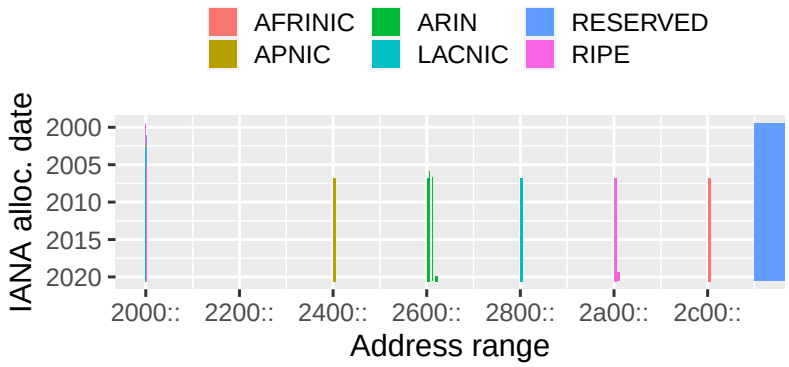
Figure 15. Visualization of the common ancestor prefix tree for a /15 subset of the region shown in Figure 14.

enterprise businesses (*e.g.,* law firms, medical groups, schools) within the metros implied by their names. Though space limitations prevent us from showing more examples here, we observed similar patterns across all selected prefixes.

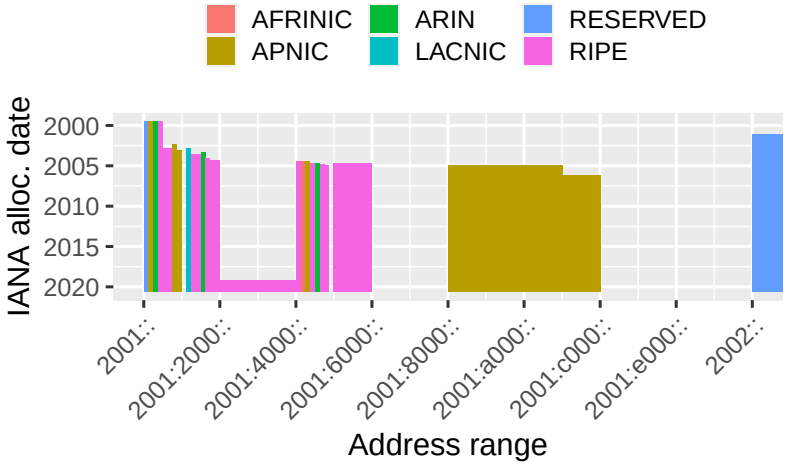
3.2.2.4 Initial population of IPv6. Similar to IPv4, IANA manages global allocations in the IPv6 address space and maintains, in particular, a registry of which global unicast IPv6 address ranges have been allocated with which RIRs[2] [12].

Figure 16 shows which particular blocks have been allocated thus far (x-axis) along with their allocation dates (y-axis) for a small region where the oldest allocations are clustered. In addition to the clear per-RIR structure with large subblocks allocated to particular RIRs (*e.g.,* APNIC was allocated all of 2001:8000::/10 in 2005) shown here, we observe that in 2006 IANA outlined large /7 blocks for each RIR (not shown) even though as of today, only a few /12s in these blocks have been allocated. We also confirmed that WHOIS IPv6 network records follow similar hierarchical organization at the RIR and organization level through analysis of∼290K

[2]Currently IANA only allocates addresses from 2000::/3 and other ranges are reserved for different purposes or future use.



(a) The non-reserved regions of the global unicast IPv6 space 2000::/3 showing how particular RIRs receive allocations in well-defined clusters.



(b) Zoomed-in view of the most densely populated cluster at the lowest addresses of the global unicast IPv6 space which includes a mix of RIRs.

Figure 16. Timeline of initial allocation of IPv6 global unicast address space from IANA to RIRs.

IPv6 records in the ARIN bulk WHOIS dataset. Overall, we conclude that the same forces that shaped IPv4 allocations are at play in IPv6 and hence similar observation about the importance of the conservative cascade model and multifractal scaling are relevant for observed IPv6 addresses as well.

3.2.3 Multifractal IPv4 Address Structure: Fait Accompli. The mathematically rigorous and empirically validated physical explanation in § 3.2.2 implies that as far as observed IPv4 addresses in measured Internet traffic are

concerned, encountering multifractal is neither a mystery nor a surprise but a "fait accompli"—a new invariant of modern Internet traffic. It results from an underlying conservative cascade model that is assumed to have generated the observed address structure in the first place and can be theoretically shown to generate multifractal measures, assuming the cascade generator satisfies certain rather weak technical conditions. Equipped with this new understanding, we fully expect that this invariant is indeed present in each of the recently collected Internet traffic traces listed in Table A.1 in Appendix A.

To confirm this expectation, we apply the statistical method of moment technique to each of these traces, mainly because this technique is known to be more robust than the geometrical histogram method, but also because the macroscopic characterization it provides for a given dataset is sufficient for making a binary inference decision; i.e., concluding that the given data is consistent with multifractal scaling behavior or it is consistent with monofractal scaling.

3.2.3.1 Inferring IP address structure with the method of moments. To apply the method of moment to a dataset that consists of the observed IPv4 addresses in a given traffic trace, we consider the measure μ_A on $[0, 1)$ that assigns to each subset $C \subset [0, 1)$ the number of observed addresses that fall in C and compute for a range of q-values the partition functions $Z(l, q)$ $(0 \leq l \leq 32)$ and the structure function $\tau(q)$ according to the step-by-step procedure described in Appendix B. Note that this procedure also outputs **generalized dimensions** D_0, D_1, and D_2.

In contrast to the microscopic quantification of multifractals provided by the multifractal spectrum, these quantities can be interpreted as providing a macroscopic quantitative assessment of observed multifractal scaling behavior. For example, D_0

is the *fractal dimension* of the support of μ_A and quantifies how much the data fills its physical support; D_1 is the *information dimension*, relates to Shannon's entropy, and provides information about how even the data density is, with higher values of D_1 indicating a more uniform density; and D_2 is related to the *correlation dimension* and measures how scattered the data is, with larger values of D_2 indicating more compactness and less scattering. As a sanity check, the procedure's output also includes $\widehat{\tau(1)}$ so we can check if it is 0 as per the definition of $\tau(1)$.

In practice, since inferring multifractal scaling behavior by means of the method of moments concerns examining large-l asymptotics of the partition functions $Z(l, q)$, we are not really interested in the behavior of $Z(l, q)$ for short prefix lengths and do not consider l-values that are smaller than 8. At the same time, given the hard upper limit of 32 (128) on the prefix lengths imposed by the finiteness of the IPv4 (IPv6) space, due to expected finite-limit effects, we also avoid considering l-values that are larger than a certain length (*e.g.,* 28 for IPv4). As a result, checking for straight line behavior in the plots of $\log Z(l, q)$ vs l in Step 2 typically involves the range of scales l between $8 \leq l \leq 28$ (for IPv4) and similar for computing the slopes of the straight lines corresponding to different values of q. Fortunately, being able to use some $15 - 20$ different scales or l-values is often sufficient to ensure that the obtained results are largely robust and not overly sensitive to common sources of error such as edge effects (i.e., not considering the coarsest and finest scales) and the choice of the range for linear fitting.

3.2.3.2 Multifractal IP address structure as a new invariant of Internet traffic.

We first apply the method of moments to the real-world dataset CAIDA-dir-A100 and the synthetically generated dataset UNIFORM100 that we used in Figure 9 to demonstrate visually the presence and absence of multifractal scaling

behavior in the dataset CAIDA-dir-A and UNIFORM, respectively. In particular, Figure 17 shows the log-log plots of the partition functions $Z(l, q)$ for integer-valued q's ($-4 \leq q \leq 4$) across the entire spectrum of possible scales $0 \leq l \leq 32$ for CAIDA-dir-A100 (left) and UNIFORM100 (right). In these figures, vertical dotted lines indicate the range of scales $8 \leq l \leq 24$ and $1 \leq l \leq 16$, respectively, used for linear fitting to estimate the structure function $\widehat{\tau(q)}$: $-4 \leq q \leq 4$ for CAIDA-Dir-A100 and UNIFORM100; the resulting structure functions are shown in the left plot (for CAIDA-Dir-A100) and right plot (for UNIFORM100) in Figure 18.

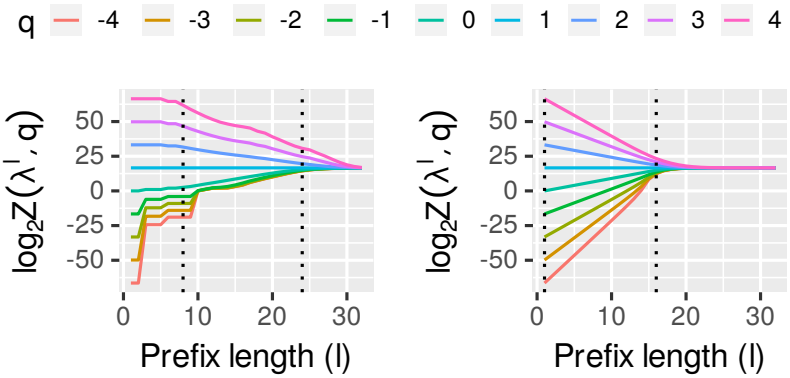


Figure 17. Partition functions for dataset CAIDA-Dir-A100 (left) and UNIFORM100 (right), respectively.

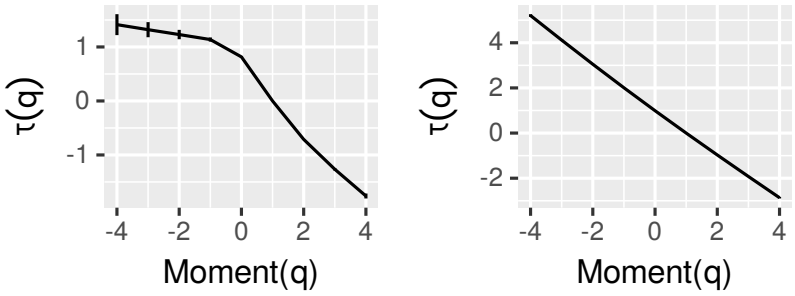


Figure 18. Structure functions for dataset CAIDA-Dir-A100 (left) and UNIFORM100 (right), respectively.

We observe that in agreement with the "visual proof" in Figure 9, Step 3.1 of the step-by-step procedure given in Appendix B implies that while UNIFORM100

is consistent with mono-fractal scaling (i.e., results in a linear structure function), the real-world dataset CAIDA-dir-A100 is consistent with multifractal scaling. Using Step 3.2 to quantify the multifractal nature of CAIDA-Dir-A100, Table 3 gives the estimates for the generalized dimensions D_0, D_1, and D_2 and also shows that the estimated $\tau(1)$-value is zero in agreement with theory. To compare, in Table 3 we also include the estimated generalized dimensions for UNIFORM100 and note that they are degenerate in the sense that $D_0 = 1$ (i.e., the data fill the unit interval); $D_1 = 0.8$ (more uniform density); $D_2 = 1$ (i.e., less scattered); and $\tau(1) = 0$. This comparison succinctly quantifies the visible differences between the top and bottom plots in Figure 9: multifractal data does not fill the physical space (i.e., $D_0 < 1$), is highly unevenly distributed (i.e., D_1 is small), and exhibits more clustering or less scattering (i.e., D_2 is large).[3]

ToI Instances	D_0 (fractal dimension)	D_1 (information dimension)	D_2 (correlation dimension)	$\tau(1) = 0$?
CAIDA100	0.8	0.6	0.6	✓
UNIFORM100	1.0	0.8	1.0	✓

Table 3. Estimated generalized dimensions.

Lastly, we repeat the above-described limited statistical analysis for each of the datasets listed in Table A.1 in Appendix A. The results of this comprehensive but purposefully limited analysis are summarized in Table A.2 in Appendix A and support our main finding that the obtained statistical evidence for a multifractal IP address structure is not limited to a specific time, place, or type of traffic. All measured traces produce partition and structure function plots (not show) that are similar to Figures 17 and 18, respectively, and yield generalized dimension estimates that are similar those obtained for CAIDA-dir-A100 and distinctly not like those obtained

[3]We also checked (not shown) that the obtained results are largely insensitive w.r.t. data size.

for RANDOM100 (see Table A.2 in Appendix A). As expected, these results are consistent with our new understanding of the root cause underlying multifractal IPv4 address structure and confirm it to be an invariant of measured traffic, a facet of behavior of measured Internet traffic which has been empirically shown to hold in a wide range of environments and include such properties as diurnal patterns of activity, (asymptotic) self-similar scaling of temporal traffic rate processes, and heavy-tailed distributions for protocol-related entities such as TCP connection or IP flow duration and size (e.g., see [60] and references therein).

CHAPTER IV

DESIGNS: TOOLS TO LEVERAGE STRUCTURE FOR TRAFFIC QUERIES

The content in § 4.1 is adapted from [118] and [116]. The content in § 4.2 is adapted from [117].

This chapter presents technical algorithms and system design components built around traffic structure characterizations obtained in Chapter III. First, we leverage the high-variance characterization of the number of distinct groups per time unit (described in § 3.1) to develop novel time-domain tools of sub-epoch sampling and scheduling in § 4.1. Next, we leverage the multifractal characterization of observed IP addresses (described in § 3.2) to develop several novel address-space-domain tools that improve on state-of-the-art prefix-level refinement algorithms in § 4.2. As discussed in § 1.4, the design elements and algorithms developed in this chapter serve as building blocks for the real-world systems constructed in Chapter V.

4.1 Temporal Tools

This section considers a class of traffic queries that group packets based on arbitrary header fields and compute per-group metrics. First, in § 4.1.1 we describe our approach to running such traffic queries in a subset of time-divisions (or *sub-epochs*) of a given time-window (or *epoch*) and how cluster sampling theory allows deriving similar accuracy bounds as for state-of-the-art sketch-based approximation methods (discussed in § 2.2.1). We then discuss in § 4.1.2 the principle algorithmic components required to translate the abstract idea of time-division approximation into a concrete telemetry system design.

4.1.1 Time-Division Approximation. At a high-level, our time-division approximation technique observes that query operations do not need to run all the time. Instead, operations can execute during strategically placed sub-windows of

the overall time window (*e.g.,* an operation could execute for 3 of 8 equal-duration sub-windows of a 5 s overall time window). Non-parametric cluster sampling theory then allows us to estimate error and future resource requirements without making assumptions about the number of observed traffic groups. In particular, § 4.1.1.1 describes how this formal connection enables a controlled tradeoff of query result accuracy for reduced resource requirements and § 4.1.1.2 describes a similar tradeoff of query result latency. After establishing these foundational designs, § 4.1.1.3 describes an extension to enable relative accuracy goals, § 4.1.1.4 describes how the same approach can apply to distinct-count queries, and § 4.1.1.5 considers how the error bounds derived for our time-division method relate to the error bounds of sketch-based methods.

4.1.1.1 Accuracy tradeoff. Given fixed scheduling epochs, time-division approximation trades off accuracy for reduced resource requirements by sampling a subset of the subepochs in which to execute a particular query. Suppose the query executes in a total of E epochs and that each epoch is divided into N equal-duration subepochs. Let $t_{i,j}$ be the query's result in the i-th subepoch of the j-th epoch, S_j be the set of which subepochs are actually sampled in the j-th epoch, $n_j = |S_j|$ be the number of subepochs sampled in the j-th epoch, and $s^2_{t_j}$ be the sample variance of the $t_{i,j}$'s in the j-th epoch. Using results from cluster sampling theory [105], the estimator

$$\hat{t}_E = \frac{1}{E} \sum_{j=1}^{E} \frac{N}{n_j} \sum_{i \in S_j} t_{i,j} \tag{4.1}$$

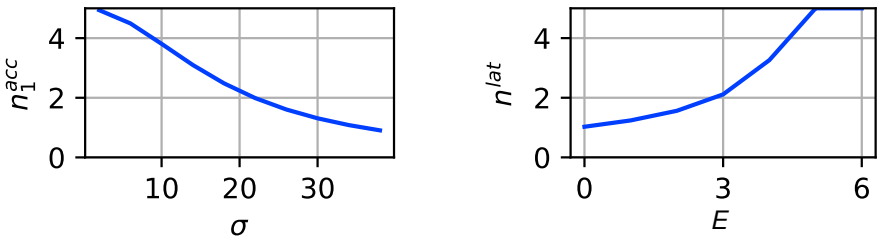
can be shown to be unbiased for the mean ($t_E = \frac{1}{E} \sum_{j=1}^{E} \sum_{i=1}^{N} t_{i,j}$) and has standard error

$$SE(\hat{t}_E) = \frac{N}{E}\sqrt{\sum_{j=1}^{E}\left(1 - \frac{n_j}{N}\right)\frac{s_{t_j}^2}{n_j}}. \tag{4.2}$$

We use Equation 4.1 to estimate query results after E epochs and Equation 4.2 to determine when accuracy goals have been fulfilled. Assuming the query has already executed in $E-1$ epochs without achieving the target standard error σ, we can rearrange Equation 4.2 as

$$n^{acc} = \frac{s_{t_E}^2 N^2}{E^2\sigma^2 - \left(\sum_{j=1}^{E-1}\text{Var}\left(\hat{t}_j\right)\right) + N s_{t_E}^2} \tag{4.3}$$

to estimate n^{acc}, the number of subepochs in which a query should execute in the E-th epoch. Note that if $\sigma = 0$, then $n^{acc} = N$ and the query will be executed in all of the subepochs in its first epoch. As σ increases, n^{acc} decreases freeing more of the subepochs for other queries. For example, Figure 19a shows the result of evaluating Equation 4.3 for the first epoch of a query, indicating that if the accepted standard error is large enough, the scheduler only needs to execute the query in a single subepoch.



(a) Increasing σ reduces n^{acc} in the first epoch.

(b) n^{lat} increases as deadline $\tilde{E} = 6$ approaches.

Figure 19. Numeric evaluations of Eqs. 4.3 and 4.4 assuming fixed variance $s_t^2 = 8$, $N = 5$, and queries get $3/5^{th}$ of the subepochs.

4.1.1.2 *Latency tradeoff.*

In addition to the accuracy tradeoff discussed above, we can tradeoff latency for reduced resource requirements by executing a query's operations across several epochs. The key observation enabling this tradeoff is that by spreading the sampled subepochs over several epochs, the query can reduce its per-epoch requirements while still attaining its accuracy goal. Network administrators leverage this tradeoff by specifying larger latency goals on queries that do not require fast returns.

Suppose a particular query has a latency goal of $\tilde{E}$ epochs. Again, assuming the query has already executed in $E-1$ epochs, we need to estimate the number of subepochs in which the query should be allocated n^{lat} in the E-th epoch with $1 \leq E \leq \tilde{E}$. First, we break the sum in Equation 4.2 into *past* ($1 \leq j < E$) and *future* ($E < j \leq \tilde{E}$) components. We then have,

$$n^{lat} = \frac{s_{t_E}^2 N^2}{\tilde{E}^2\sigma^2 - N^2\left(past + future\right) + N s_{t_E}^2}. \tag{4.4}$$

While the *past* component can be calculated directly using observations from prior epochs, the $future$ component must be estimated based on the number of subepochs the query expects to receive in future epochs. Administrators can tune this expected number of subepochs based on current and expected query workloads. Figure 19b shows the result of evaluating Equation 4.4 in each epoch leading up to a query's target latency of $\tilde{E} = 6$ assuming that the operation gets $3/5^{th}$ of the number of subepochs requested in each epoch. Since in this case, the query is not given its full requested number of subepochs, the target n^{lat} increases dynamically to meet the deadline indicating that Equation 4.4 can dynamically drive scheduling decisions even when its results are not taken literally in each epoch (as may be the case when multiple queries compete for resources).

4.1.1.3 Approximation based on relative error goals.

A key challenge with the formulation described in Equations 4.3 and 4.4 is that $s^2_{t_j}$ (and hence $SE(\hat{t}_E)$) varies based on several aspects of traffic (*e.g.,* burstiness) and system parameters (*e.g.,* epoch duration). As a result, it is often challenging to determine an appropriate value of target standard error σ before a query is run. We address this challenge for the common scenario where network administrators submit the same (or similar) query (queries) in contiguous epochs by accepting *relative* accuracy goals which we express using the coefficient of variation, [1] call the *target coefficient of variation*, and denote by c_v. For queries submitted with c_v (instead of σ) as their accuracy goal, we maintain an internal estimated target standard error $\hat{\sigma}$. When making scheduling decisions before each epoch, we uses the current value of $\hat{\sigma}$ in place of σ in the same methods described previously in this section and in § 4.1.2. After each epoch, compute a new target standard error $\hat{\sigma}' = c_v \cdot \hat{t}_E$ and updates $\hat{\sigma}$ to follow the new target using EWMA (*i.e.,* $\hat{\sigma}_{new} = \alpha \cdot \hat{\sigma}' + (1-\alpha) \cdot \hat{\sigma}_{old}$). Note that, although we could have directly adapted Equations 4.1-4.4 using the definition of c_v, we found that doing so translated the inter-epoch burstiness of the point estimate $\hat{t}_E$ into n^{acc} and n^{lat}, making them too unstable to drive consistent scheduling decisions. The EWMA-based approach smooths over local burstiness while still removing dependence of the target accuracy goal on the relative magnitude of query results.

4.1.1.4 Correcting distinct operators.

While the previous sections discuss foundations for making sound approximations of packet/byte counts, many useful queries also involve identifying and counting distinct elements. To correct estimates for a common class of such distinct queries (including several queries in Table 1), we leverage the Chao estimator without replacement [45, 44]. The intuition

[1] Coefficient of variation is a standard statistic defined as the ratio of standard deviation to mean.

behind the Chao estimator is that the number of rare elements in the sample (*e.g.,* the number of flow keys observed exactly once or twice across all subepochs of a query) give an approximation of the number of rare elements in the underlying traffic and hence can be used to remove the bias induced by flow keys missed due to sampling. Similar to the cluster sampling estimators described earlier in this section, the Chao estimator can be used to obtain point and standard error estimates based only on the observed samples.

4.1.1.5 Comparison with sketch-based methods. As discussed in § 2.2.1, state-of-the-art telemetry systems primarily rely on sketch-based approximation. However, we argue that the cluster sampling method described in this section is preferred for situations where traffic composition may change in unexpected and unpredictable ways. Consider again the scenario shown in Figure 6 where the number of distinct sources observed increases suddenly. Suppose another telemetry system was using count-min sketch [50] (which computes similar point estimates as the approach described in this section) in the same scenario. The accuracy of query results produced by count-min sketch is given by $\hat{t}_i < t_i + \varepsilon||t||_1$ where $\hat{t}_i$ is the estimated query result for the i-th distinct element (source-destination pair in the example), t_i is the ground-truth query result, $||t||_1 = \sum_i |t_i|$ is the ground-truth L1-norm of all observed elements, and ε is a constant based on the number of sketch counters allocated. When the number of sources observed increases, $||t||_1$ also increases proportionally loosening the upper bound on $\hat{t}_i$. However, $||t||_1$ is a ground-truth value (depending on t_i, *not* the estimate $\hat{t}_i$) which must be estimated offline and cannot easily be extracted from the sketch counters. As a result, the network administrator would receive no indication from the telemetry system that the error of results may be critically compromised. On the other hand, in our approach each sampled subepoch will reflect the increased

number of sources and estimated query results (Equation 4.1) and result accuracy (Equation 4.2) will continue to accurately reflect observed traffic.

4.1.2 Subepoch-Level Scheduling. This section describes concrete optimization-based scheduling techniques that utilize the formalizations introduced in § 4.1.1 to decide which query operations should run in each sub-epoch. We first present core elements of the scheduler in § 4.1.2.1, then in § 4.1.2.2 consider several additional scheduling policies to deal with inherent challenges in using optimization frameworks directly.

4.1.2.1 Optimization Formulation. We cast the task of generating query schedules as an optimization problem and adapt well-known techniques to generate schedules through this casting. We apply our optimization formulation every epoch to determine which queries should execute in each of the N subepochs as shown in Algorithm 1. First, in line 2 we use the DISENTANGLE method of Yuan et al. [179] to break the submitted queries Q into disjoint traffic slices K (based on their filter predicates) and save the mapping between queries and slices in $s_{i,k}$. Line 3 then computes the minimum number of stateful update operations required by the reduce operators of all queries in each particular slice. These steps ensure that, even when the filter predicates of multiple submitted queries overlap, we can use combined update operations in U and disjoint traffic slices in K to compute all queries on a single switch hardware stage. Next, lines 4 through 6 compute estimates of the memory and subepoch requirements of each query. Finally line 7 creates and solves the optimization problem described below. If a feasible solution cannot be found, line 9 falls back to a heuristic scheduling method described in our Supplementary Materials.

Algorithm 1 Method for determining subepoch schedule

```
1: procedure GET-SCHEDULE(Q, u, SE)
2:     K, s ← DISENTANGLE(Q)
3:     U ← COMBINE-UPDATES(u, K, s)
4:     m ← ESTIMATE-MEMORY
5:     n^acc ← EQUATION 4.3(σ)
6:     n^lat ← EQUATION 4.4(σ, E)
7:     d ← SOLVE-OPTIMIZATION
8:     if d is infeasible then
9:         d ← GET-HEURISTIC-SCHEDULE
10:    end if
11:    return d
12: end procedure
```

Inputs. Table 4 shows the particular inputs and outputs of this optimization problem. Of the input variables, t_k, u_i, $s_{i,k}$, T, A, and M are known exactly based on submitted query requirements and available switch resources, while m_i, n_i^{acc}, and n_i^{lat} must be estimated based on observation of past epochs. Our current implementation uses EWMA to estimate m_i and $s^2_{t_E}$ (as required by n_i^{acc} and n_i^{lat}) independently for all update operation types. We leave exploration of more sophisticated estimation approaches to future work. Scheduling decisions are encoded in the $d_{i,j}$ indicator variables which determine which queries should execute in each subepoch. We do not consider the division of switch memory between queries since memory is dynamically allocated during the aggregation operation (see § 5.1.1.5).

Constraints. We impose the constraints shown in Table 5 to satisfy two high-level requirements: *(i)* respecting switch resource limits (C1, C2, C3) and *(ii)* forcing minimal progress in each query and ensuring variance estimates are well-defined (C4). Note that C2 captures the fact that if two queries rely on the same update operation, they can be merged to use a single ALU. In the case that the estimated quantity m_i

Variable	Description
Q	index set of queries ready for execution
SE	index set of subepochs
K	index set of all disjoint traffic slices
U_k	index set of all update operations in slice k
t_k	number of TCAM entries required by slice k
u_i	index of update operation required by query i
$s_{i,k}$	indicator that query i requires slice k
m_i	memory required in each subepoch by query i
n_i^{acc}	number of subepochs required for accuracy goal for query i (§ 4.1.1.1)
n_i^{lat}	number of subepochs required for latency goal for query i (§ 4.1.1.2)
T	total available TCAM entries
A	total number of available switch ALUs
M	total available SRAM counters
$d_{i,j}$	indicator that query i executes in subepoch j

Table 4. Variables used in optimization formulation of scheduling problem. The sole outputs $d_{i,j}$ determine the schedule for the next epoch.

$$\textbf{C1:} \quad \forall j \in SE,\ \sum_{k \in K} t_k \mathbf{I}\left[\bigvee_{i \in Q} d_{i,j} s_{i,k} = 1\right] \leq T$$

$$\textbf{C2:} \quad \forall j \in SE,\ k \in K,\ \sum_{u \in U_k} \mathbf{I}\left[\bigvee_{i \in Q} d_{i,j} s_{i,k} \mathbf{I}\left[u_i = u\right] = 1\right] \leq A$$

$$\textbf{C3:} \quad \forall j \in SE,\ \sum_{i \in Q} d_{i,j} m_i \leq M$$

$$\textbf{C4:} \quad \forall i \in Q,\ \sum_{j \in SE} d_{i,j} \geq 2$$

Table 5. Scheduling problem constraints to respect (C1) TCAM capacity requirement, (C2) switch ALU capacity, (C3) SRAM capacity, and (C4) query minimal progress requirement. $\mathbf{I}\,[]$ is the indicator function.

$$\textbf{O1:} \quad \text{minimize} \sum_{i \in Q} \left| \left(\sum_{j \in SE} d_{i,j} \right) - n_i^{acc} \right|$$

$$\textbf{O2:} \quad \text{minimize} \sum_{i \in Q} \left| \left(\sum_{j \in SE} d_{i,j} \right) - n_i^{lat} \right|$$

$$\textbf{O3:} \quad \text{minimize} \sum_{i \in Q, j \in SE} d_{i,j} m_i$$

Table 6. Objective functions considered in the multi-objective formulation.

turns out to be violated by traffic conditions in the subsequent epoch, we simply drop new aggregation groups once the available switch memory is totally consumed.

Objectives. In computing the schedule of each epoch, we consider the objective functions listed in Table 6. O1 seeks to satisfy accuracy goals by minimizing the distance to the value of n^{acc} computed in Equation 4.3, O2 seeks to satisfy latency goals by minimizing the distance to the value of n^{lat} computed in Equation 4.4, and O3 seeks to limit the maximum volume of data that needs to be returned from the switch in a single subepoch. We expose the Pareto front of these objective functions using linear scalarization which allows administrators to express the importance of each objective by submitting weights and is computationally efficient.

Problem shape and size. Note that the only variables that are solved during evaluation of the optimization problem are the $d_{i,j}$ which determine which queries execute in each subepoch. Also, the number of constraints is linear in the maximum of the number of subepochs and the number of queries. For example, if 100 queries are to be executed in 8 subepochs, the resulting optimization problem has 800 binary variables and 124 constraints. Overall, the on-line optimization problems in our method are much smaller and simpler compared to the off-line optimization problems considered in, *e.g.,* Sonata [70] (since Sonata considers the product of all possible query partitionings and different prefix-level refinement plans). We observe that off-the-shelf optimization solvers (*e.g.,* [6]) are able to solve our problems in a number

of milliseconds (compared to the 20 minute time limit set on the optimization solver in Sonata) making their use in our on-line system feasible.

Additionally, since the number of queries ready for execution in each epoch is given by the particular query arrival process, the only system parameter that impacts problem size is the number of subepochs per epoch. Intuitively, configuring our approach to run with more subepochs per epoch exposes more opportunities in the optimization problem to multiplex larger numbers of queries on the given processing resources. However, the number of subepochs is directly linked with both epoch duration and subepoch duration (in particular epoch duration is subepoch duration times number of subepochs). Assuming network administrators fixed epoch duration based on their particular monitoring requirements (since epoch duration fixes minimum latency across all queries), adding more subepochs also leads to shorter subepochs reducing the fraction of time spent actually monitoring traffic compared to the fixed amount of time required to reconfigure switch hardware between each subepoch. In light of these facts, the number of subepochs (or, equivalently, subepoch duration) must be configured carefully to avoid either too constrained optimization problems (too few subepochs) or too much reconfiguration overhead (too many subepochs). We provide an empirical illustration of this tuning requirement in § 5.1.3.5 which demonstrates that between 4 and 8 subepochs per epoch achieves a sort of sweet spot between these two extremes regardless of epoch duration.

4.1.2.2 Challenges of Online Optimization. Unlike prior work (*e.g.,* [70]), the inputs to our optimization problem are dependent on task dynamics (*e.g.,* the set Q can vary each epoch) and traffic dynamics (*e.g.,* the suggested n_i^{acc} could increase in response to increased traffic variability). Hence, we must solve the optimization problem independently in each epoch. However, although our problems

are usually simple enough to be solved in a number of milliseconds as mentioned above, invoking an optimization solver in an online scheduling method is still fraught with challenges. First, certain combinations of inputs and constraints can lead to infeasible problems where it is impossible to satisfy all constraints. Second, since integer programming is a well known NP-complete problem, finding an optimal solution can take exponential time in the worst case. In what follows, we describe several precautions that we take in the design of our scheduler to ensure these challenges do not adversely affect the performance of the telemetry system.

Dealing with infeasible queries. Our first strategy to deal with infeasible optimization problems is to require that all submitted queries can be executed on the given switch resources in the absence of other queries. In particular, if a query requires more than T TCAM entries, A ALUs, or M counters, the scheduler must reject that query outright, since it will not be able to execute on the given switch hardware. This ensures that our scheduler can always make progress on the current pool of submitted queries by selecting a single query and allocating the full switch resources for all subepochs. We note that a query partition scheme similar to Sonata [70] could be added to our system to handle this case more elegantly, but leave this to future work.

Dealing with slow optimizations. To deal with the potentially exponential time that could be required to converge to an optimal solution, we limit the duration of time spent in the optimization algorithm to an acceptable fraction of total epoch time. This method, known as early stopping, is a well-known technique to gather feasible, good, if not fully optimal solutions [131, 133]. When the optimization process stops due to this time limit, the current solution must still be checked for feasibility and only allowed to execute if it is, in fact, feasible.

Fail-safe. In cases where the optimization problem is either proven infeasible or times out before converging, we fall back to a simple heuristic "fail-safe" mode of scheduling. We also deny all new query submissions when in fail-safe mode to notify network administrators that the system is currently saturated and to prevent the accumulation of a large backlog which could cause the optimization problem to remain infeasible in future epochs. Our simple heuristic fail-safe scheduling scheme greedily selects the query closest to its deadline and allocates this query fully to switch resources. To increase progress in fail-safe mode, we also add other queries that use the same or a subset of the selected query's traffic slices until either the memory or ALU limit is reached. Since queries scheduled in this mode execute for each subepoch, $n_j/N = 1$ for that epoch ensuring progress towards accuracy targets (*i.e.,* that the standard error $SE(\hat{t}_E)$ decreases as E increases: see Equation 4.2), though some queries may suffer increased latency.

Another approach to dealing with situations where a feasible schedule cannot be found is to send slices of traffic to the collector and compute query results in software. In this approach queries running during fail-safe mode could still meet tight latency goals at the expense of increased load on the collector. Depending on the nature of situation triggering fail-safe mode, this could impose infeasible processing loads on the collector or lead to excessive congestion between switch and collector. In future work, we plan to investigate solutions to this problem including combinations of heuristic scheduling and moving query operations to software.

4.2 Spatial Tools

This section considers a class of traffic queries that seek to detect particular sets of addresses (*e.g.,* addresses responsible for sending volumetric DDoS attack traffic). Rather than monitoring all addresses directly, which may be infeasible due

to limited SRAM, our approach leverages prefix-level metric (or feature) collection and iterative refinement. We first describe how our novel "scheduling" approach to prefix-level refinement "zooms-in" on the target prefixes in § 4.2.1, then describe two key algorithmic components to improve the efficiency and robustness of prefix-level refinement in § 4.2.2.

4.2.1 Iterative Prefix Refinement. To describe our approach to prefix-level refinement, we first consider limitations of several baseline approaches (§ 4.2.1.1), then introduce several key policies of our design (§ 4.2.1.2), and finally consider how to leverage a result of multifractal scaling of observed addresses (see § 3.2)—that there is no "ideal" prefix length—by dynamically deciding at what prefix length to terminate refinement (§ 4.2.1.3).

4.2.1.1 Baseline Approaches. In general, approaches to prefix-level iterative refinement [120, 179] maintain a list of monitored prefixes F. Packets are grouped by these prefixes and aggregate features computed for each prefix (*e.g.,* by submitting dataflow queries [70, 187, 118]) during a monitoring window or *epoch* (*e.g.,* 1 s). Between epochs a set of decisions are made about how to update F for the next epoch based on features computed in the previous epoch. Typically these decisions are to "zoom-in" on a particular prefix (*e.g.,* replace 10.10.0.0/16 with 10.10.0.0/17 and 10.10.8.0/17) or to "zoom-out" by combining two sibling prefixes. In the case of volumetric DDoS signature detection, the goal is to zoom-in on prefixes that contain attack sources and zoom-out on benign prefixes.

Existing approaches to prefix-based refinement have one of two undesirable properties. First, approaches like MRT [179] use tree-like data structures to implement prefix matching on CPUs and zoom-in on every suspected attack prefix, leading to an exponential increase in the number of prefixes whose features must be

collected and processed during each epoch. Second, approaches like DREAM [120] use a fixed number of TCAM entries on programmable switches and every time they zoom in on a prefix, they also choose another pair of siblings to zoom out on.

Our initial experiments quickly confirmed that neither of these approaches are sufficient for volumetric DDoS attack signature detection. The MRT approach of zooming in on every prefix classified as suspicious quickly exhausts the limited number of TCAM slots. The proposed algorithms in DREAM [120] also struggle to effectively zoom in to sufficiently long prefix lengths. Consider the simple example where there is only one attack prefix, 10.0.0.0/8. As shown in Figure 20, DREAM requires monitoring eight other prefixes even though they may be entirely empty.

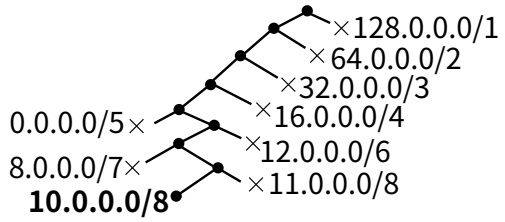


Figure 20. Monitoring one attack prefix 10.0.0.0/8 in DREAM [120] requires monitoring an additional 8 benign or empty prefixes in order to maintain a complete cover of the address space.

4.2.1.2 Scheduling Prefixes on Fixed Monitoring Slots.

To address these algorithmic short-comings in the light of volumetric DDoS attack signature detection, we develop a novel approach based on the high-level idea of scheduling which prefixes to monitor each epoch. We augment the set of prefixes F with two other sets: R which contains prefixes which have already been reported as part of the attack signature, and H which contains non-scheduled prefixes "held-out" in CPU memory for consideration in future epochs. During each epoch, we use a fixed number of prefix monitoring slots compiled into switch hardware to compute features for prefixes in F (each slot computes all features for a single prefix). Between epochs,

we apply a procedure called EpochUpdate() to update all three sets to zoom in on attack signatures. The key decisions of our approach stem from two high-level scheduling policies, *children-first* and *never-zoom-out* as illustrated in Figure 21.

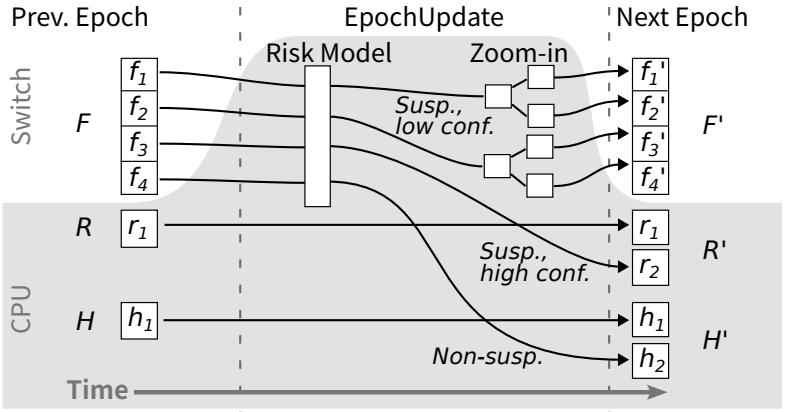


Figure 21. Iterative prefix refinement to zoom-in on suspicious prefixes while using fixed monitoring resources.

Children-first. The children-first policy is based on the intuition that if the model classifies a prefix as containing attack traffic, it is more likely that a child of that prefix will also contain attack traffic in the next epoch compared to any other prefix. As a result, we schedule all newly-zoomed-in-on children before any other prefixes under consideration. Only once these child prefixes have been cleared as non-suspicious, do we remove them from F to H.

Never-zoom-out. The never-zoom-out policy is based on the intuition that even if a prefix does not look suspicious in a particular epoch, it may still become suspicious in the future (*e.g.,* due to changes in attack source, or fluctuations in benign traffic). As a result, instead of zooming out as in DREAM, we collect all prefixes ever zoomed in on in H. This way, even if the attack vectors or sources change, our approach can quickly recall its previous progress and continue zooming in where it left off, thus

avoiding the slow control loop vulnerability [24] associated with approaches that do not maintain this kind of longer-term state (*e.g.,* Jaqen [103]).

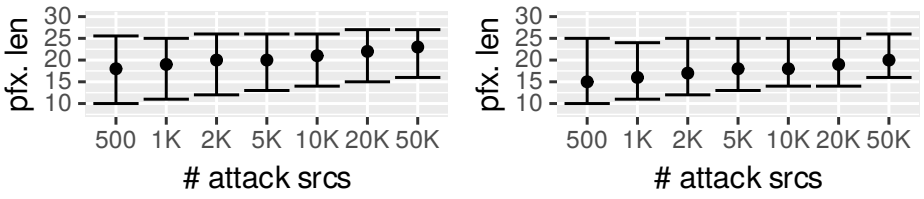
Note that although F is constant in size, $R \cup H$ grows after most epochs during the active-attack phase. We leave questions of how to interpret and ultimately reset F, R, and H in the post-attack phase and/or how to optimally pre-condition these sets in the pre-attack phase to future work.

Key parameters. Our approach includes two key parameters that effect the iterative refinement process. First, `prefixesPerEpoch` determines the number of feature monitoring slots available in switch hardware (*i.e.,* an upper bound on $|F|$). Increasing `prefixesPerEpoch` enables our approach to observe a larger region of the address space each epoch and can in some cases reduce the number of epochs to reach an accurate attack signature, but increases our approaches hardware resource footprint. Second, `bitsPerEpoch` determines how many bits are added to the length of prefixes when we zooms in on them. Although intuitively larger value of `bitsPerEpoch` allow us to zoom-in faster, since each zoom-in decision generates `2^bitsPerEpoch` children, setting `bitsPerEpoch` too high generates too many children to monitor and can actually cause our approach to zoom-in slower due to the limit imposed by `prefixesPerEpoch`.

4.2.1.3 Deciding Length of Reported Prefixes.

Given that prefix lengths increase monotonically in our approach (by the never-zoom-out policy), a key consideration is when a prefix flagged by the model as potentially containing attack traffic should be reported in the attack signature. On the one hand, reporting at too short of prefix lengths leads to high false-positives. On the other hand, reporting at too long of prefix lengths leads to wasting monitoring resources and increase detection

time. Ultimately, as shown in Figure 22 optimal prefix-level partitions of attack and benign sources require a wide range of prefix lengths (*e.g.,* /10 to /25).



(a) Mirai sources (b) Uniform random sources

Figure 22. Distribution of lengths of prefixes used for optimal separation between attack and benign traffic in realistic attack scenarios with varying numbers of attack sources. (See § C.1 for details of the scenarios.)

To better approximate these optimal prefix-level partitions, we develop an "early stopping" method to decide when prefixes that are flagged by the model should be included in the report set R (and hence removed from active consideration in F and H). Since the primary concern is to avoid reporting prefixes that also contain benign sources, early stopping leverages a profile of benign traffic collected during the pre-attack phase.

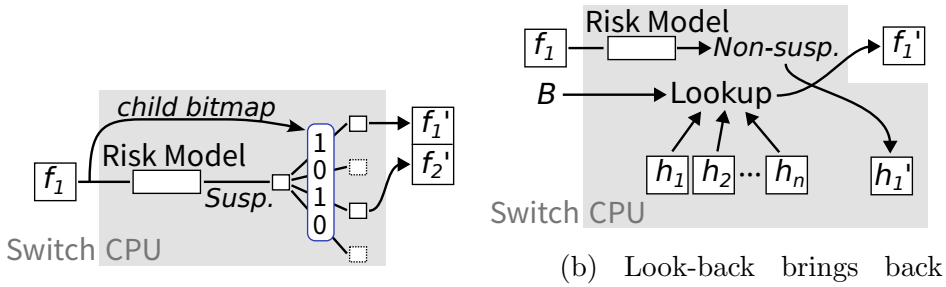
In particular, we define $\textsc{Benign-Proximity}(y, p) = n/2^{32-p}$ for a given prefix y of length p where n is the number of benign sources observed in y during the pre-attack phase. During the active-attack phase, if the model flags a prefix f of length ℓ_f as suspicious, we report f and move it to R if and only if $\textsc{Benign-Proximity}(f, \ell_f)$ is less than a threshold. Computing $\textsc{Benign-Proximity}(y, p)$ requires monitoring distinct sources for the most recent m seconds in the pre-attack phase. In our evaluation, we found that m=120 s yields sufficiently accurate results while requiring modest resources. For example, in our hardware prototype we used a ∼131 KB Bloom filter.

4.2.2 Improving Refinement Speed and Robustness. Although the children-first and never-zoom-out policies are key to realizing our refinement techniques, they require additional algorithmic components (beyond those described in § 4.2.1) to cope with dynamic changes to the set of target address (*e.g.,* changes to attack sources observed in modern volumetric DDoS attacks). We first discuss our "look-ahead" technique which reduces the number of prefixes which must be monitored when "zooming-in" (§ 4.2.2.1), then discuss our "look-back" technique which keeps a low-overhead view of regions of the address space not currently scheduled for monitoring (§ 4.2.2.2).

4.2.2.1 Look-Ahead to Avoid Empty Children. The children-first policy requires allocating monitoring slots for all children of any prefix flagged by the model as suspicious but not yet ready for reporting. However, due to the relative sparsity of observed addresses, some of these children almost always turn out to be inactive and hence waste precious switch resources. For example, with `bitsPerEpoch` = 4 and a parent prefix with only one active child, we would waste 15/16 monitoring slots.

To address this problem, we develop a novel *look-ahead* method that encodes information about which children are active in the features gathered for each parent prefix. In particular, as shown in Figure 23a we add a per-prefix "child bitmap" where each bit represents a potentially active child. When a packet matches prefix f of length ℓ_f, we extract the ℓ_f + `bitsPerEpoch` bits of that packet's source address and use them as an index into f's child bitmap. During EpochUpdate(), if f is identified as suspicious by the model, we then read f's child bitmap and only select the children of f whose bits were set for monitoring in the next epoch. The child bitmap requires adding an extra feature with 2^`bitsPerEpoch` bits per prefix. For

example, at `bitsPerEpoch` = 4, this only adds 16 bits (compared to the several 32 fields already required for the other features).



(a) Look-ahead increases zoom-in rate by focusing on active children.

(b) Look-back brings back active hold-out prefixes when attack changes.

Figure 23. Extensions to the scheme of Figure 21 to deal with dynamic changes in the set of attack sources.

4.2.2.2 Look-Back to Catch Changes. Although the never-zoom-out policy ensures that our approach can always make progress towards refining attack signatures even when the attack sources change, a critical consideration not addressed in § 4.2.1 is which prefixes from H should be added to F when extra monitoring slots are available. Again, due to relatively sparse population of the observed address space, simplistic methods like taking the first `prefixesPerEpoch` from H lead to wasting monitoring slots on empty prefixes.

To address this problem, we develop a *look-back* method that casts a wide net over all regions of the address space not monitored in F. Our key observation is that we do not necessarily need to discover exact regions sourcing new attack traffic, but only need to re-focus the refinement process on currently active regions of the address space. In particular, as shown in Figure 23b we use a simple Bloom filter [37] B to build an (approximate) list of distinct sources that don't match prefixes in F. Then when extra monitoring slots are available, we select prefixes from H based on their membership in B.

CHAPTER V

APPLICATIONS: ALGORITHMS IN ACTION

The content in § 5.1 is adapted from [118] and [116]. The content in § 5.2 is adapted from [117].

This chapter describes how we combine characterizations of traffic structure from Chapter III and design elements from Chapter IV into practical telemetry systems targeting real-world traffic monitoring tasks. First, in § 5.1 we describe a system named DynATOS, which leverages our time-domain designs to process large numbers of dynamically-submitted traffic queries while remaining robust in the face of the time-domain structure of traffic. Second, in § 5.2 we describe a system named ZAPDOS, which leverages our space-domain designs to quickly detect prefix-level signatures of volumetric DDoS attacks.

5.1 Handling Query and Traffic Dynamics with DynATOS

This sections describes how we leverage the temporal design elements and algorithms introduced in § 4.1 to develop a concrete telemetry system for dynamically-submitted traffic queries that remains robust under realistic time-domain traffic structure. In particular, § 5.1.1 describes the concrete design decisions made to realize DynATOS, § 5.1.2 describes our implementation and evaluation of a prototype version of DynATOS build on real programmable switch hardware, and § 5.1.3 describes an extended evaluation of the performance of DynATOS based on simulation over real-world packet traces.

5.1.1 DynATOS System Design.

5.1.1.1 Overview. At its core, DynATOS is composed of three main components as shown in Figure 24. Network administrators submit queries along with accuracy and latency goals to the scheduler via a high-level REST API. The scheduler

then translates queries into their primitive operations and constructs schedules for how these operations should be run on switch hardware given a stateful awareness of current traffic compositions based on observed results of previously executed queries. These schedules are then handed to a (state-less) runtime component which communicates with switch hardware to execute the primitive operations and collect intermediate results. Once ready, the runtime component gathers all results and passes them back to the scheduler and network administrators.

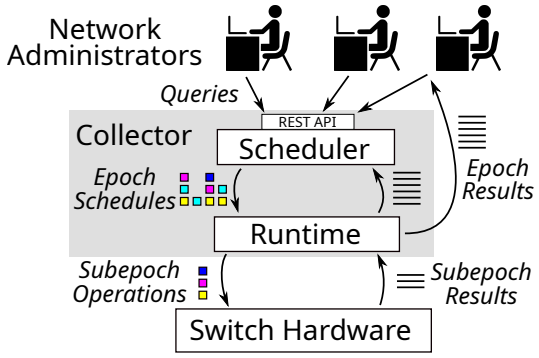


Figure 24. Architecture of DynATOS.

Following the designs outlined in § 4.1, accuracy goals in DynATOS can be expressed either in relative or absolute terms. Relative accuracy goals are specified using target coefficient of variation (CV) and absolute accuracy goals are measured using target standard error. From a network administrator's perspective, using the relative accuracy measured by CV may be a more reasonable and practical choice because CV is independent of the magnitude of the underlying values computed, that is, it is "unit-less" or "dimension-less". By basing accuracy goals on CV and automatically converting to standard error based on current traffic conditions, DynATOS frees network administrators from a potentially painful accuracy goal tuning process. For example, a network administrator can simply specify an accuracy

goal by setting a CV target of ±10% for all queries submitted and then let DynATOS automatically find the corresponding standard error over several epochs.

5.1.1.2 Scheduling horizon. Since queries can arrive at any time, we must decide when and for how far into the future resources should be scheduled. We first examine several possible approaches to this problem, then describe our approach in the next paragraph. One option is to compute the schedule each time a new query arrives and adjust all existing queries to the new schedule. While this option minimizes the time a query has to wait before it can start executing, it complicates the realization of accuracy and latency goals since the duration of the scheduling horizon (*i.e.,* until the next query arrives) is unknown when forming the schedule. Alternatively, we could compute the new schedule each time all queries in the prior schedule terminate. While this option ensures schedules can be executed exactly as planned, newly submitted queries may experience a longer delay.

We choose, instead, to make scheduling decisions at fixed windows of time which we call *epochs* (*e.g.,* every 5 s). This allows a balance between the two schemes mentioned above: queries must wait at most the duration of one epoch before executing and during an epoch queries are ensured to execute according to the schedule. In particular, we divide the scheduling epoch into N *subepochs* and our scheduler assigns subsets of the submitted queries to each subepoch as shown in Figure 25. Subepochs provide flexibility to schedule different queries at different times while also providing concrete resource allocation units. Queries submitted during an epoch are checked for feasibility and only considered in the following epoch. For example, in the figure, Q4 is added sometime during epoch 2, but cannot be scheduled until epoch 3. During the epoch, the scheduler collects intermediate results for each subepoch in which a query is executed and aggregates these subepoch results based

on the query's aggregation operation. Once an epoch completes, results of complete queries are returned, while new and incomplete queries (*e.g.,* queries that have not yet met their accuracy goal, or queries that have a longer latency goal) are considered for the next epoch. For example, in Figure 25 Q3 completes execution in the second subepoch of epoch 2 and its results are returned during the scheduler invocation before epoch 3. We further assume that each query executes over traffic in a single epoch and telemetry tasks requiring longer measurement durations than our scheduling epoch can simply re-submit queries.

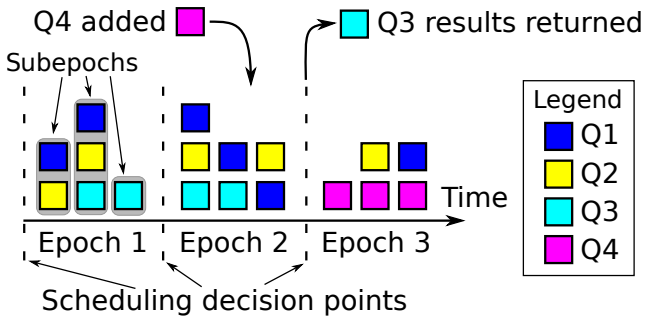


Figure 25. Example of scheduling 4 queries with $N = 3$ subepochs per epoch.

5.1.1.3 Design Challenges. In order to successfully leverage the ideas introduced in § 5.1.1.1, we must solve several concrete design challenges discussed below.

D1: Approximating generic query results. Efforts like Marple and Sonata develop expressive query description languages which map into data plane computation models. However, approximation of query operations is often necessary due to limited data plane resources and massive traffic volumes. It is unclear how state-of-the-art approximation methods can be leveraged to work with queries expressed in languages like Marple or Sonata. As discussed in § 2.2.1, the currently proposed baseline approach of simply replacing stateful reductions in Sonata queries

with sketch-based primitives requires prior knowledge of worse-case traffic conditions and does not perform well under dynamic traffic scenarios.

On the other hand, directly setting a fixed number of subepochs in which to run query operations as discussed in § 5.1.1.1 does not consistently translate into query result accuracy. To illustrate, Figure 26 shows the accuracy (as F1 score) of four queries on a single trace with a fixed allocation of 6 out of 8 subepochs (details of the queries, metrics, and settings used in this experiment are given in § 5.1.2). The accuracy of each query is clearly different with median over the trace ranging from ∼0.48 for TNC to ∼0.93 for DDoS demonstrating the challenge in reasoning about query accuracy from the number of subepochs in which the query is executed.

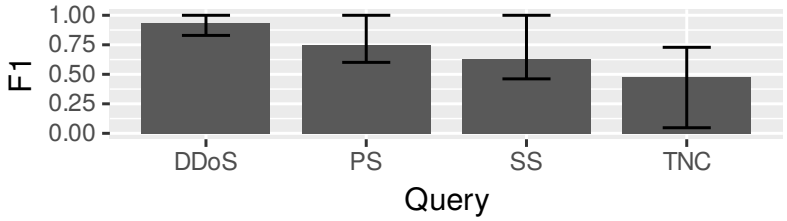


Figure 26. Median F1 score of various queries on a single trace (MAWILab [61] 2022-08-30) under fixed allocation of 6 out of 8 subepochs in all epochs. The different accuracy achieved by each query illustrates the challenge of translating the number of subepochs executed to query accuracy.

D2: Estimating accuracy of approximations. Approximate query results must be accompanied with a sound estimate of their accuracy. This is critical for network administrators to understand the system's confidence in detecting a particular event or reporting a particular metric and equally critical in dynamic telemetry systems to inform the balance of resources between approximate queries. Prior efforts have made progress towards this goal [120, 121, 78], but none anticipate accuracy estimation for current state-of-the-art generic query descriptions in dynamic telemetry systems.

We extend the illustration of Figure 26 by considering a single port scan (PS) query run in a fixed number of subepochs. Each line in Figure 27 shows distribution of accuracy achieved by PS (run with the corresponding number of subepochs) over a sample of 28 traces from MAWILab [61]. The wide spread of each distribution on the x-axis indicates that query accuracy varies significantly in response to different traffic compositions in the sampled traces, regardless of number of subepochs. Even when all 8 subepochs are allocated, the query still achieves less than perfect accuracy on some traces due to the brief system down times during reconfiguration between subepochs. As a result, even if the relationship between number of subepochs and accuracy could be captured statically for individual queries, query accuracy would still vary greatly depending on the underlying traffic composition.

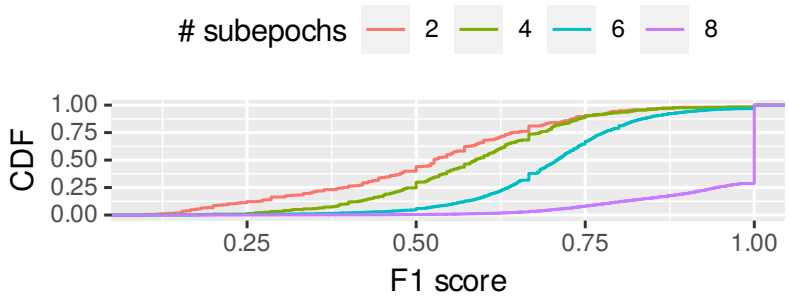


Figure 27. Distribution of F1 score (w.r.t. ground truth) of the port scan query for different (fixed) numbers of subepochs over sample of 28 traces from MAWILab. For each fixed number of subepochs, the query achieves a wide range of F1 scores over different traces implying the system must dynamically estimate accuracy.

D3: Allocating finite hardware resources among variable sets of queries under traffic dynamics. Very few prior efforts address the need of a telemetry system to evaluate multiple concurrent queries on finite hardware resources. In order to handle traffic dynamics, such a system must dynamically update resource allocations based on the estimated accuracy of each query. Moreover, since it is

possible that the given resources will be insufficient to meet the accuracy of all queries, such a system must enable network administrators to express query priorities and allocate resources with respect to these priorities.

5.1.1.4 Our Solutions. We develop a novel approximation method based on cluster sampling theory and runtime programmable capabilities to address D1 and D2. Cluster sampling is known to be a good fit for scenarios where the overheads (*e.g.,* cost) of sampling large groups of the population (*e.g.,* subepochs) are significantly lower than the overheads of sampling individual population members (*e.g.,* packets) [105]. Runtime programmability exposes exactly such a scenario: large groups of packets can be sampled each subepoch and only (small) aggregate results need to be reported to the collector. In contrast, per-packet sampling, where each packet needs to be considered a candidate for sampling, incurs non-trivial per-packet overheads (in switch hardware) and a copy of each sampled packet needs to be sent to the collector resulting in non-trivial communication overheads.

We leverage cluster sampling to address D1 by applying it to the first aggregation operator in multistage queries. For example, in the DDoS query we only approximate computation of the distinct source-destination pairs list and execute all subsequent operations exactly. The intuition behind this is that each aggregation operator in a telemetry query reduces the volume of data passed to the next operator. Therefore, reducing the resource requirements and volume of data emitted from the first aggregation reduces the load on all subsequent operators.

Cluster sampling naturally addresses D2. As the underlying traffic composition changes, each sampled subepoch presents a snapshot of the changed traffic composition. § 4.1.1 describes the details of how we use the formal error bounds of cluster sampling to actively adapt resource allocations as traffic composition changes

across multiple epochs. Compare this with sketch-based traffic monitoring. Because a sketch's accuracy degrades as the number of keys increases, it is difficult to tell from the sketch counters alone if the traffic composition has changed. Moreover, without knowing the ground truth traffic composition, it is difficult to determine the accuracy of a sketch-based result in order to drive allocation decisions in future epochs.

To address D3, we integrate our approximation technique in a scheduler that determines how a number of concurrent queries should be executed on a single switch hardware, balancing resources between queries to satisfy accuracy and latency goals set by network administrators. As described in § 4.1.2, our scheduler uses a novel multi-objective optimization formulation of the problem of when to run which queries given query priorities and resource constraints. This formulation allows the scheduler to balance between the goals of multiple concurrent queries, sometimes allocating less than the exact number of subepochs when queries have lower priority and resources are scarce (*e.g.,* due to a large number of concurrent queries).

Finally, we develop a runtime system leveraging these ideas to efficiently execute schedules on switch hardware, gather intermediate results, apply factors to correct for sampling, and return results to network administrators in a high-level format. Based on these results, administrators can then decide to execute new queries in the subsequent epoch, or to re-execute the same queries.

To illustrate how our key ideas apply to telemetry system design, we return to the examples introduced in Table 1. First, cluster sampling method can be applied to the first aggregation stage of a wide range of queries including the DDoS, port scanning, and TCP latency queries. Second, if the number of sources increase (*e.g.,* as shown in Figure 6), each sampled subepoch in our system will return a proportionally increased number of tuples thereby increasing the volume of reports exported while maintaining

high accuracy. Compare this to sketches where the increased number of sources would increase the number of hash collisions reducing accuracy of results. Note that state-of-the-art approaches [189, 188] dynamically allocate new sketches when the number of keys changes (to preserve sketch accuracy). However, this approach is infeasible in current switch hardware. Finally, if the network administrator needs to dig into root causes (*e.g.,* of TCP latency), our scheduling approach handles the burst of queries by adapting the optimization problem to account for the new query's accuracy and latency goals while assigning query operations to limited switch hardware.

5.1.1.5 Limitations and Assumptions. **Monitoring problems addressed by DynATOS.** As shown in Figure 28, DynATOS can monitor traffic queries whose first steps (in the Sonata [70] paradigm) are filter, key-by, reduce followed by arbitrary post-processing. In the current work we only apply approximation to the first three operators (*i.e.,* to the first aggregation) and compute post-processing exactly. Moreover, DynATOS's approximation method implies the traffic features computed by these queries satisfy the following assumptions.

- Feature values do not fluctuate excessively over measurement durations of one or two seconds.
- The monitoring task can be implemented using features gathered at a single point in the network.
- Features are constructed from packet header fields and/or other switch-parsable regions of the packet.
- Features can be computed using atomic filter, map, and reduce operations.

Under these assumptions monitoring tasks like detecting microbursts [47], identifying global icebergs [63], and detecting patterns in TCP payloads [38] cannot be efficiently executed using DynATOS. However, tasks like the DDoS, port-scanning, and latency

detection examples of Table 1 along with a wide range of tasks considered in prior efforts with similar assumptions (*e.g.,* [120, 123, 70]) can be effectively executed using DynATOS.

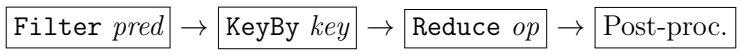


Figure 28. Visualization of type of queries supported by DynATOS as a pipeline of atomic operations.

Switch hardware model. In the following, we assume a restricted runtime programmable switch hardware model. In this model, switch hardware is able to execute the first `Filter`, `KeyBy`, and `Reduce` operators shown in Figure 28 for a number of independent queries maintaining dynamic allocation of per-key state between queries. Similar to Newton [187], our switch hardware allows arbitrary parameterization of these operators *at runtime*. For example, switch hardware could execute the filter and reduce commands required by the Sonata TCP new connections queries for a period of time, then quickly (*e.g.,* within a few milliseconds) be re-programmed to execute the filter and reduce commands required by the Sonata DDoS query. We note that our scheduling methods are independent of this particular switch hardware model and could readily be applied to more fully programmable ASICs [39, 8].

Network-wide scheduling. Ultimately, administrators need to query traffic across different logical or physical domains of their network. This implies that telemetry systems should collect information from a distributed set of switches (or other monitoring points) and provide a global view of network traffic. In this work, we consider only a single monitoring point (*e.g.,* a critical border switch) and leave the challenges of distributed scheduling of telemetry operations to future work.

Nonetheless, a single switch deployment on a enterprise or data center border switch can still be highly effective in executing the types of queries considered.

5.1.2 Hardware Prototype Evaluation. In this section, we describe our evaluation of DynATOS and demonstrate the following key results.

- The time-division approximation technique in DynATOS is more robust than state-of-the-art in the face of traffic dynamics and offers comparable performance to state-of-the-art sketch-based approximate techniques (§ 5.1.2.2).
- The scheduling method in DynATOS handles dynamic query workloads with up to one query every second and leverages specific accuracy and latency goals to reduce per-query resource usage (§ 5.1.2.3).
- Latency overheads in DynATOS are minimal and dependent on the load on the collector and the number of queries which must be updated in switch hardware (§ 5.1.2.4).

***5.1.2.1 Experimental Setup.* Setting.** We evaluate DynATOS on a BCM 56470 series [18] System Verification Kit (SVK) switch running BroadScan [2] which implements the telemetry operations described in § 5.1.1.5. Our version of BroadScan has $A = 8$ parallel ALU operators, and a flow table with $M \approx 9$MB of memory. A software agent on the switch's CPU manages reconfiguration of hardware in response to requests from the collector. Our collector and scheduling software runs on a server with an Intel Xeon Gold 5218 CPU at 2.3Ghz and 383GB memory. This server is equipped with a 40Gb Mellanox MT27700-family network card connected directly to the SVK's data plane. A separate 10Gb Intel X550T network card on the same server connects to the SVK's management interface to manage updates to hardware configuration as schedules execute.

Traces. Unless otherwise stated, we replay a trace from the MAWILab traffic data set (Sept. 1st, 2019) [61] using `tcpreplay` [17]. We selected this trace as a baseline because some of its features are static while others are more dynamic.

Default parameters. We use five-second scheduling epochs to allow sufficient measurement duration without incurring excessive delay of results which must wait for epoch boundaries. We divide epochs into $N = 8$ subepochs so that the schedule has sufficient options for arranging queries without making subepochs too short to generate useful samples. We set objective weights to balance between priorities and suppose queries will get all future subepochs when evaluating Equation 4.4. Queries are submitted with realistic values of σ based on baseline measurements of their variances in the trace. We set $\alpha = 1/2$ in the EWMA estimation described in § 4.1.2.1. Bars show median and error bars show 5^{th} and 95^{th} percentiles over all epochs of the trace.

Query workloads. We use DynATOS to implement four of the telemetry queries originally introduced by Sonata [70] and used in several recent efforts. Our hardware model handles a fixed sequence of filter and reduction operations so we implement the remaining query operations in software. This scenario is equivalent to Sonata with a limited number of switch hardware stages. We report the accuracy of approximate implementations of these queries as F1 score (the harmonic mean of precision and recall) by comparing against ground truth computed offline. In addition to static queries, we generate dynamic query workloads based on random processes to evaluate DynATOS (see § 5.1.2.3). To the best of our knowledge, there is no comparable publicly-available dynamic query workload benchmark. Our workloads are publicly released at [13] to support validation of our results and to facilitate benchmarking of similar systems in the future.

Implementation. We implement the DynATOS scheduler in ∼14k lines of C and C++. Following ProgME [179], we use BDDs to represent query filter conditions in our implementation of the DISENTANGLE algorithm (§ 4.1.2.1). We use the open source CBC implementation [6] to solve the optimization problems described in § 4.1.2.1. Our implementation also defers some result processing operations to the time spent waiting for results from switch hardware to improve efficiency.

Comparisons. We compare DynATOS with ElasticSketch [176], Newton [187], and SketchLearn [78]. We modified the implementations of both ElasticSketch and SketchLearn to support the filter and reduce operations required by several of the Sonata [70] queries. Though we were unable to locate a publicly available implementation of Newton, we implemented its sketch-based approach to approximating Sonata's primitive operators. In particular, we use count-min sketch [50] to approximate the `reduce` operator and a bloom filter [65] to approximate the `distinct` operator.

5.1.2.2 Performance of Time-Division Approximation. Robustness in the face of traffic dynamics.

To address the question of what happens when traffic composition changes significantly we consider an excerpt from the MAWILab dataset taken on Nov. 14th, 2015. As shown in Figure 6, this excerpt features nominally static traffic followed by a dramatic surge in the number of sources around 100 seconds into the trace.

To understand how different methods handle this change in traffic dynamics, we first tune each method's parameters to achieve high accuracy (F1 ¿ 0.9) on the first 100 seconds of the excerpt, then run the method with these parameters over the entire excerpt. Since it is possible that this anomaly was cause by some form of DDoS attack, we run the DDoS query in this scenario to locate the victim of the attack.

This is intended to reflect a realistic situation where a method was deployed and tuned for a particular traffic composition, which then changes. In real deployments, such changes could be caused by attacks or performance anomalies and represent the moments when data collected from a telemetry system is most critical.

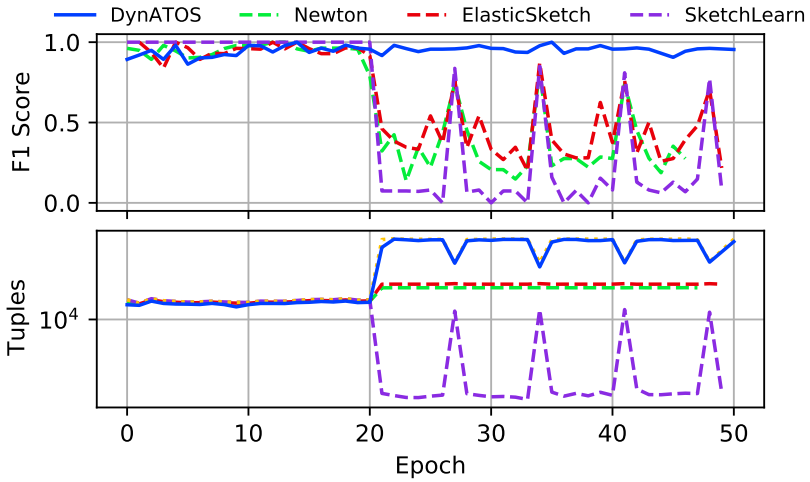


Figure 29. Performance of different methods on the 2015 MAWILab excerpt shown in Figure 6.

Figure 29 shows the F1 score and number of tuples returned to the collector in each epoch over the trace excerpt. All methods achieve high accuracy for the first 20 epochs, but then when the number of sources increases after the 20^{th} epoch, they diverge significantly. First, we note that DynATOS is able to maintain high accuracy where other methods suffer by dynamically increasing the load on the collector. This is a result of the natural robustness of our non-parametric sampling method: when the underlying traffic composition changes, those changes are reflected in each sampled subepoch causing the volume of data reported for each subepoch to increase to ensure steady accuracy.

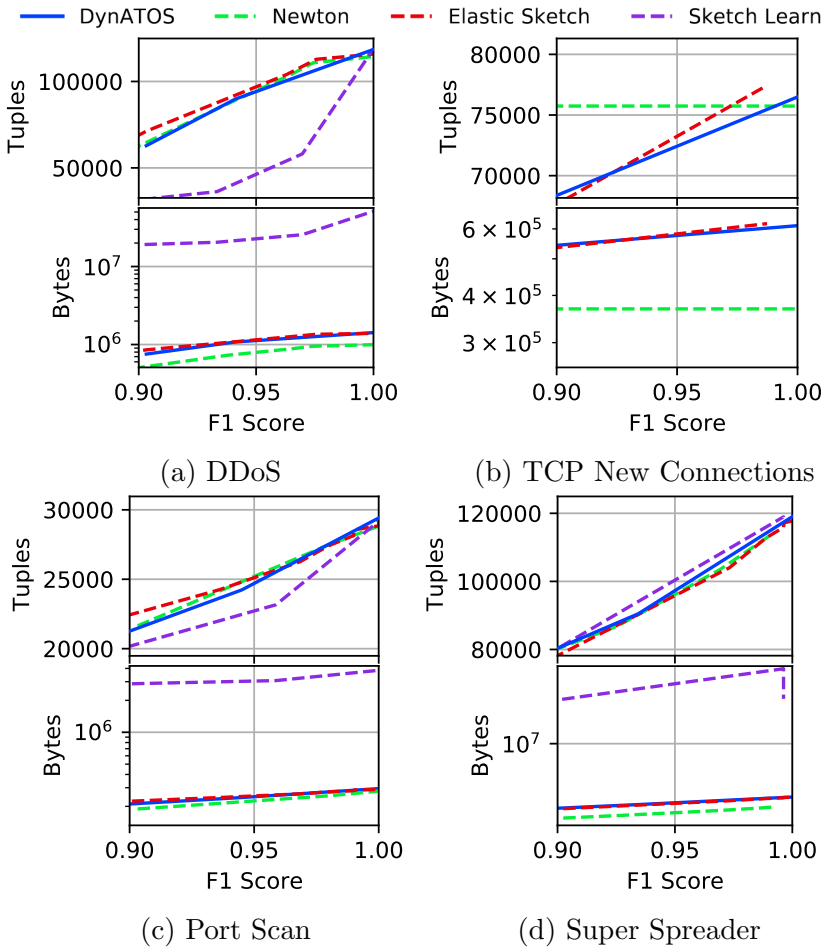
The sketch-based methods in ElasticSketch and Newton, on the other hand, are limited by the static table sizes configured for the first 20 epochs: once the traffic composition changes, these tables become saturated and excessive hash collisions lead

to F1 scores below 0.5. We confirm that the average number of hash collisions per epoch jumps by 2× when the traffic distribution changes in epoch 21. We note that these sketch-based methods also offer no easy way to estimate the accuracy of returned results, so while an operator may become suspicious due to the slight increase in load on the collector, they would have no way to verify that the accuracy of these methods is compromised.

Sketchlearn differs from other methods in that it reconstructs flow keys based on data stored in a multi-level sketch. Sketchlearn guarantees only that it will be able to extract all flows that make up more than $1/c$ of the total traffic where c is the fixed number of columns in the sketch. We confirm that in this trace, the increased number of sources is caused by a large number of small flows (one to two packets). As such, the threshold to be extracted increases, but none of the added flows are able to meet it and so SketchLearn is unable to extract existing as well as new flows with high enough confidence. SketchLearn does associate accuracy estimates with these results so an operator could be notified of this situation, but would have to reload their switch's pipeline with a larger value of c in order to achieve acceptable accuracy.

Overall accuracy-load tradeoff. As in previous efforts [70], we consider the volume of data returned from switch hardware to the collector (*i.e.,* load on the collector) as a critical resource. Each approximation method can reduce this load while reducing accuracy of query results, leading to a performance curve in accuracy vs. load space. To empirically estimate this curve, we determine several different parameterizations of each method, execute the method with each parameterization over all epochs of the trace, then compute the accuracy and load on collector in each epoch. For some queries the sketch-based methods must export their full sketches to the collector so we report load in terms of both tuples (the number of records or events) and bytes

(the total size of data). We use the median of each value over all epochs to estimate the empirical performance curves.



(a) DDoS

(b) TCP New Connections

(c) Port Scan

(d) Super Spreader

Figure 30. Empirically measured accuracy vs. overhead curves of DynATOS and sketch-based alternatives for several queries from Table 1.

Figure 30 shows performance curves for four different queries with two plots for each query showing overhead as tuples and bytes on the y-axis. Here we use the baseline MAWILab trace so these results represent a mostly static traffic scenario. Note that the lower right-hand corner of these plots is ideal with maximal accuracy and minimal load. We observe that DynATOS' novel approximation method (§ 4.1.1) performs as well as, if not better than other methods. The sketch-based method

proposed by Newton achieves slightly better performance in terms of total data volume on the DDoS and Super Spreader queries because it only sends flow keys from the first distinct operator whereas other methods also return a counter. SketchLearn requires relatively large multi-level sketches to be exported each epoch in order to achieve comparable accuracy on these queries despite it's lower tuple counts. In the case of TCP new connections, we were unable to run a large enough sketch to reach the accuracy range shown here for other methods. We observe that for the TCP new connections query Newton's count-min sketch is highly sensitive to sketch size. For example, adding a single additional counter moves the F1 score across the entire range shown in the plot. DynATOS, on the other hand, achieves comparable if not higher performance and offers a wider range of load savings.

5.1.2.3 Performance of Scheduling Algorithm. **Dynamic query workload.** Real telemetry system deployments must deal with dynamics in the number and types of queries submitted to the network over time. Since, to the best of our knowledge, no representative dynamic query workloads are available, we synthesize such workloads based on the following scheme. First, we generate a series of base queries with random aggregation keys and granularities and arrival times based on a Poisson process with rate λ. We suppose these base queries are submitted by a human operator or automated process which then submits followup queries based on base query results. In particular, when each base query terminates, we submit between 0 and 3 followup queries with the same aggregation as the base query, but filters added to select a single aggregation group from the base query's results. For example, if a base query with aggregation key source IP address at 8 bit granularity returned results for 0.0.0.0/8, 10.0.0.0/8, and 192.0.0.0/8, we might submit followup queries to monitor just 10.0.0.0/8 and 192.0.0.0/8. To provide contrasting accuracy

and latency goals, base queries are submitted with looser accuracy goals ($\sigma = 100$) and latency goals randomly chosen within a range of 1 to 5 epochs, while followup queries are submitted with tighter accuracy goals ($\sigma = 50$) and a latency goal of 1 epoch.

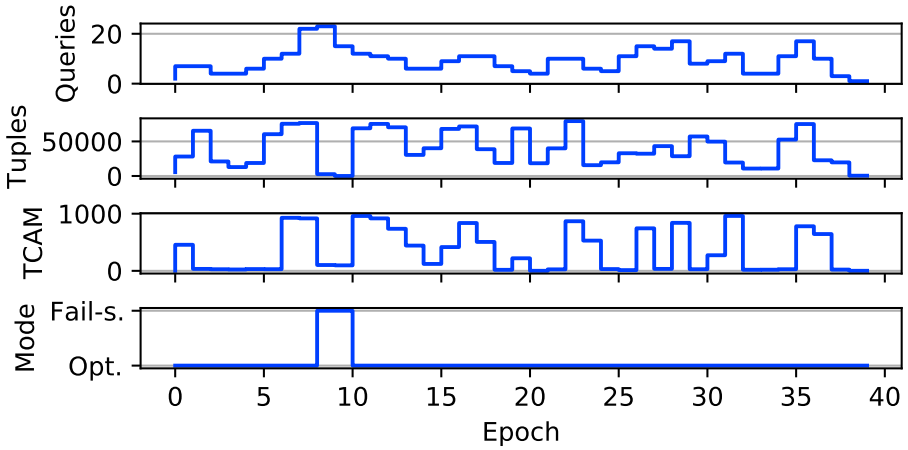


Figure 31. Example time-series of a dynamic query workload (3/5 queries per second).

Figure 31 shows the evolution of the number of queries submitted by one of our dynamic query workloads (top plot) and traces of different operating metrics (lower three plots). In this workload, the maximum number of queries is submitted in epoch 8 which leads to an infeasible schedule since too many TCAM entries are required to keep track of all filter groups of followup queries. This causes our scheduler to enter fail-safe mode for two epochs to dispatch with the excess queries. Note that the heuristic algorithm currently used to select queries in fail-safe mode only selects a few queries based on fully disjoint traffic slices leading to reduction of load on collector and TCAM utilization. Under the software-based fail-safe mode mentioned in § 4.1.2.2, the load on collector would continue increasing here while TCAM utilization would drop.

To understand how DynATOS scales with the rate of dynamic query workloads, we generate a set of five workloads with different base query arrival rates. Figure 32 shows how these different workload intensities affect the performance of DynATOS in terms of queries served (Queries), tuples emitted to the collector (Tuples), TCAM entries used (TCAM), epochs spent in fail-safe mode (% Fail-s.), and the percentage of satisfied queries (% Sat.) all per-epoch. We count the number of queries satisfied as the total number of queries that received valid results during the workload run. Note that some queries submitted when the scheduler is in fail-safe mode are denied at submission time allowing an operator to re-submit these queries later. In these experiments we observe that all successfully submitted queries receive results within their target accuracy and latency goals.

We observe that, as expected, the number of queries serviced, load on collector, and number of TCAM entries required all scale linearly with the base query rate. As also expected, the number of queries satisfied decreases as more epochs are spent in fail-safe mode. We observe that the main contributor to infeasible scheduling problems in this scenario is the number of TCAM entries required to satisfy followup queries' filter conditions. We plan to investigate integration of more efficient TCAM allocation algorithms in future work to address this bottleneck.

Relaxation of accuracy & latency goals. Next, we evaluate how our approximation and scheduling method is able to reduce the per-query resource requirements in response to relaxed accuracy and latency goals. We execute the TCP new connections query with varying accuracy and latency goals and measure resource usage over 10 epochs at each setting. Here we report ALU-seconds and counter-seconds which combine both the number of ALUs (or counters) used by the query and the duration for which these resources were used.

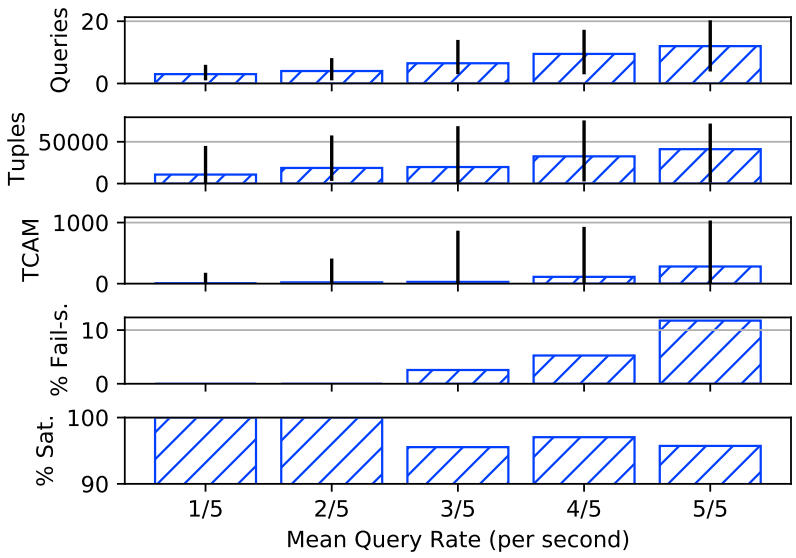
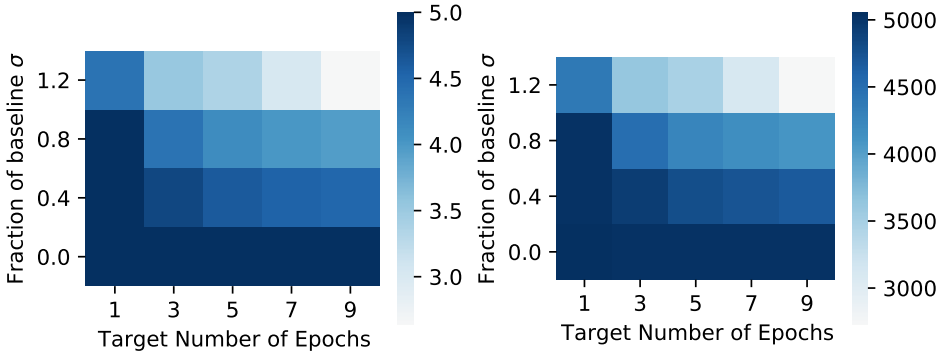


Figure 32. Performance of DynATOS on dynamic query workloads.



(a) ALU-seconds (b) counter-seconds

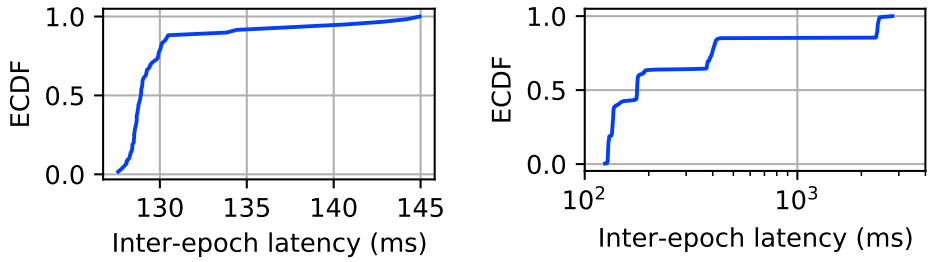
Figure 33. Evaluation of median resource usages for selected accuracy (y-axis) and latency (x-axis) targets for a single query. Lighter colors indicate lower resource usages.

Figure 33 show the resulting resource usages as both accuracy and latency goals vary in the form of heatmaps. These results demonstrate that both accuracy and latency goals can help DynATOS leverage our time-division approximation method to reduce resource requirements.

5.1.2.4 Scheduling loop overheads.

Closed-loop systems like DynATOS must quickly gather results and update switch hardware configurations between each subepoch in order to avoid missing potentially critical traffic. We define the inter-epoch latency as the total time spent not waiting for results from switch hardware. In other words, the inter-epoch latency is the total time taken by our system to gather results, reconfigure hardware operations, and decide which operations to execute in the next epoch. We observe two distinct factors that contribute to the inter-epoch latency: the load on the collector and the number of queries installed in switch hardware.

Latency vs. load on collector. The first factor contributing to inter-epoch latency is the volume of data that must be returned and processed after each subepoch. To isolate this effect, we generate synthetic traffic consisting of a certain number of sources each sending a steady stream of packets controlled by a Poisson process. We then run a query that returns a single record for each source so that by varying the number of sources in the traffic, we directly control the number of records returned and hence the load on collector.



(a) 10 Records (b) 10^5 Records

Figure 34. Distribution of inter-epoch latency in our DynATOS prototype for different loads on the collector.

Figure 34 shows the distribution of total latency for two different loads. We observe that the median inter-epoch latency in both cases is less than 130 ms, but

that with higher load the tail latencies grow to over a second. This is likely due to that fact that the collector code must allocate larger memory blocks to process the increased number of tuples returned from the switch. We leave a full investigation of the performance of our software collector to future work.

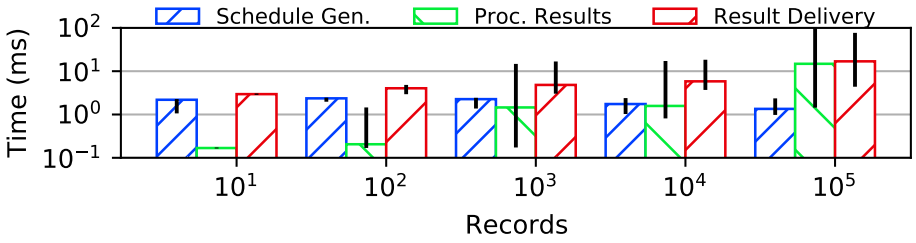


Figure 35. Software overheads in our DynATOS prototype as function of tuples exported.

We further investigate how the different components of our query scheduler impact this overall inter-epoch latency by instrumenting the scheduler. Figure 35 shows the latency break down as a function of the number of records processed for three key components: the time to generate a schedule for the epoch (Schedule Gen.), the time spent processing intermediate results at the end of the epoch (Proc. Results), and the time spent sending results back to the query-submitting process (Result Delivery). The results demonstrate that the main variable software latency is the time to process results which scales nearly linearly with the number of records. A more significant bottleneck is imposed by the result delivery time due to the use of a simple REST protocol which requires new TCP connections and data marshaling via JSON. We leave exploration of more efficient IPC mechanisms for this interface to future work.

Latency vs. number of queries. The second main factor contributing to inter-epoch latency is the time required to install and remove query operations on switch hardware. This factor is influenced primarily by the amount of state which must be

written into hardware memory which is a function of the number of queries to be installed or removed. We generate synthetic workloads containing different numbers of disjoint queries based again on the TCP new connections query and instrument our switch agent to measure the time taken by writes into hardware memory.

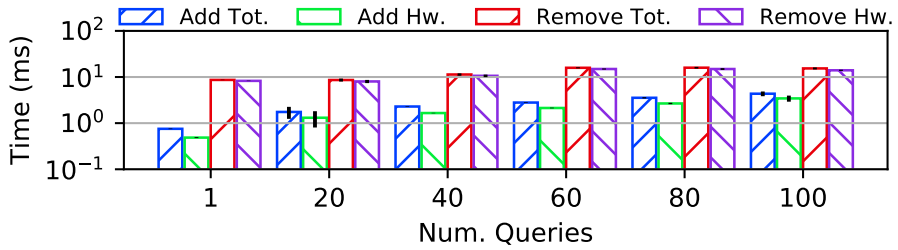


Figure 36. Hardware overheads in our DynATOS prototype as function of number of queries.

Figure 36 shows the time taken by the hardware writes to add and remove operations (Add Hw. and Remove Hw.) as well as the total time taken by the switch agent (Add Tot. and Remove Tot.) which includes the time to deserialize and validate configurations sent from the collector. These results show that up to 100 queries can be added or removed on our prototype in ∼10 ms (comparable to latencies reported in prior efforts [120, 187]). We also observe that the deserialization and validation conducted by the switch agent imposes minimal overhead. Finally, the total contribution of switch hardware to the overall inter-epoch latency is dominated by operation removal. This is because when removing operations, the switch agent must also reset the entire flow table used by these operations so as to avoid future operations anomalously reporting leftover results.

5.1.3 Simulation-Based Evaluation.

5.1.3.1 Experimental Setup. **Setting.** We extend our previous evaluation in this work using packet-level simulation.

Default parameters. All default parameters are the same as in § 5.1.2 with the additional parameter of target CV set to $c_v = 5\%$ by default.

Query workloads. As shown in Table 1, we use DynATOS to implement four of the telemetry queries originally introduced by Sonata [70] and used in several recent efforts. We report the accuracy of approximate implementations of these queries as F1 score (the harmonic mean of precision and recall) by comparing against ground truth computed offline. In addition to static queries, we generate dynamic query workloads based on random processes to evaluate DynATOS. To simulate workloads with different levels of bursty query arrivals, we use a fractional Poisson process [95, 130] to generate query arrival times. Fractional Poisson processes generalize the classic Poisson process by adding a parameter μ which controls the spread of the inter-arrival distribution. When $\mu = 1$ the fractional Poisson process converges to the classic Poisson process. As μ goes to zero, the inter-arrival distribution spreads out inducing long-term dependencies or burstiness in the query arrival rate. To illustrate, Figure 37 shows synthetically generated fractional Poisson processes with the same mean rate, but different values of μ. We normalize query arrival times so that all workloads have a mean query arrival rate of 1 query per second over a 900 s workload duration. Our workloads are publicly released at [13] to support validation of our results and to facilitate benchmarking of similar systems in the future.

Traces. To understand how DynATOS performs on a wider variety of representative traffic, we took a simple random sample of 28 days from MAWILab's [61] 2015 dataset. Each day consists of a 15 min trace starting at 2 pm. To illustrate changes in traffic composition, we compute the CV of number of keys per-epoch and number of packets per-epoch for each trace. Figure 7 shows the distribution of CVs over all traces in our sample. The distribution of key-based CVs (Figure 7a) and count-based CVs

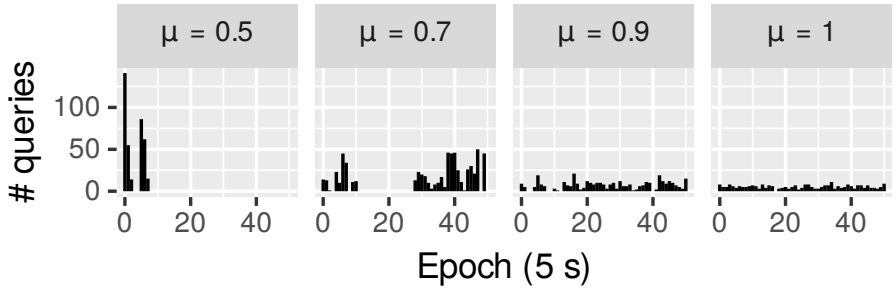


Figure 37. Number of queries submitted each 5-second epoch for example fractional Poisson query arrival processes with different "burstiness" parameter μ. (Only first 50 epochs are shown for clarity.)

(Figure 7b) are wider for 2015 compared to other recent years indicating these traces provide higher diversity in traffic composition for evaluating DynATOS.

Implementation. The simulation used in this section is built on the same software components implemented for our hardware prototype system (see § 5.1.2). At a high-level the only modification we make is to substitute a separate software module which implements the same interfaces as the switch hardware controller. This software module also simulates the impact of time spent reconfiguring hardware by dropping r seconds worth of traffic each time it is re-programmed. Based on our previous evaluation of hardware latency overheads [118], we set r to be 10 ms.

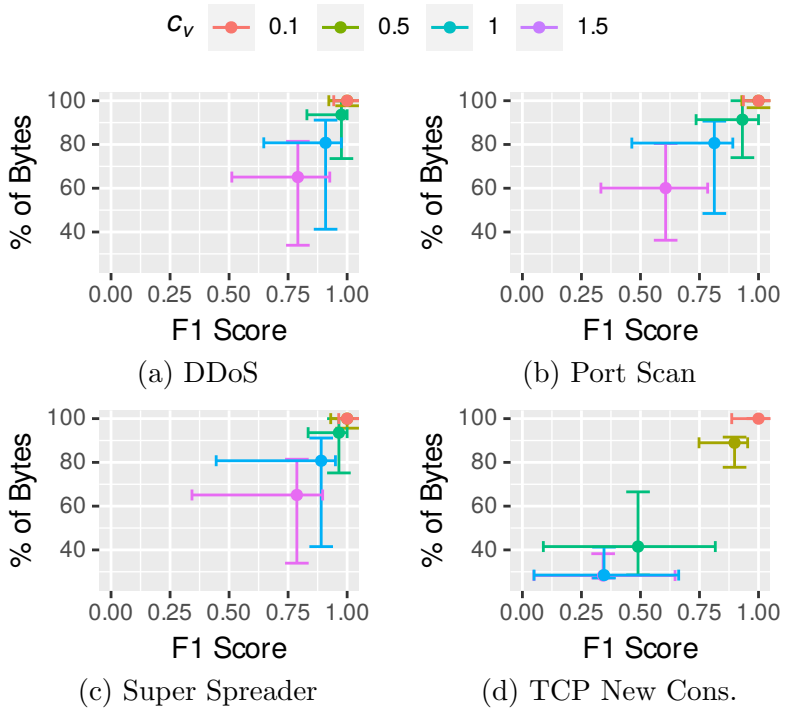
5.1.3.2 Impact of Traffic Dynamics. To evaluate the impact of traffic dynamics, we run each query from Table 1 over each of the 28 traces sampled from MAWILab. To demonstrate the tradeoff between accuracy (F1 score) and load on collector (tuples, bytes) each query is run over each trace for four settings of target CV $c_v \in \{0.1, 0.5, 1.0, 1.5\}$.

Figure 38 shows load on collector as a percentage of total number of bytes required to compute ground truth (y-axis) against F1 score (x-axis) for each setting of c_v summarized over all 28 traces. With the exception of the TCP new connections

query, all queries achieve similar accuracy ranges for each c_v value. For example, at $c_v = 1.5$ DDoS and super spreader achieve median F1 score of ∼0.9 while port scan achieves median F1 score of ∼0.8 for ∼20% median reduction in bytes sent to the collector. As shown in Figure 7a, the underlying CV of the number of keys observed in each epoch is much higher for TCP new connections (median of ∼84%), likely a product of how this query only looks at SYN packets and each SYN packet is typically associated with a new flow. As a result, the relationship between c_v and F1 score is *quantitatively* different, though *qualitatively* follows a similar pattern as for the other queries. For example, at $c_v = 0.5$ TCP new connections achieves median F1 score of ∼0.9 and an ∼11% reduction in bytes sent to the collector. Overall, as a rule-of-thumb, we note that for distinct-count queries (*e.g.,* DDoS, Port Scan, and Super Spreader), c_v of up to 1.5 results in reasonable accuracy whereas for count-based queries like TCP new connections, c_v should be kept lower (*e.g.,* up to 0.5).

Given that Figure 38 shows a relatively large range of F1 scores over all traces for each particular c_v, we further investigate how different properties of the underlying traces impact F1 score. In particular, we compute the CV of the number of keys in each epoch across each trace in our sample as a metric to summarize the trace's level of traffic composition dynamics.

Figure 39 compares the level of dynamics in each trace (x-axis) with the F1 score achieved by DynATOS for a fixed setting of $c_v = 0.1$ (we observe qualitatively similar trends for other values of c_v). As expected, different queries observe different levels of traffic composition dynamics (*e.g.,* CV between 0.03 and 0.42 for DDoS compared to 0.38 and 1.15 for TCP new connections). We note that for most queries, F1 score is only weakly correlated with underlying trace dynamics demonstrating DynATOS's ability to provide consistent accuracy in the face of trace dynamics. The TCP new



(a) DDoS (b) Port Scan

(c) Super Spreader (d) TCP New Cons.

Figure 38. Performance tradeoff for single queries over sample of 28 15-minute MAWILab traces for different target CV goals (colors show different target c_v).

connections query is again a bit of an exception for similar reasons as in Figure 38: since each key is typically only associated with a single SYN packet, when the number of keys varies, DynATOS has little opportunity to catch keys that arrived during unsampled subepochs.

5.1.3.3 Impact of Query Workload Dynamics. To evaluate the impact of query workload dynamics on DynATOS, we run dynamic query workloads generated by a fractional Poisson arrival process as described in § 5.1.3.1. To minimize the impact of trace dynamics, we use a single trace from our sample with relatively low CV across all query key types (in particular we use the trace from Feb. 22). To compare the two options for specifying accuracy goals, we use DynATOS+ to refer to results based on relative accuracy goals and DynATOS to refer to results based on absolute accuracy goals (recall § 4.1.1.3). We configure the relative accuracy queries

(a) DDoS (b) Port Scan

(c) Super Spreader (d) TCP New Cons.

Figure 39. Performance for single queries at a fixed CV goal (0.1). Each point compares F1 score and CV of the number of keys per epoch (relative to each particular query type) for a single trace from our sample.

(DynATOS+) to target $c_v = 0.1$ and compare against absolute accuracy queries (DynATOS) with fixed σ set based on baseline measurements of observed standard deviation in the trace as in [118].

Figure 40 shows query satisfaction (defined as the fraction of queries in the workload that achieve standard error less than σ) over 10 independent query workloads generated at each setting of arrival burstiness parameter μ. In addition to query workload satisfaction (top), we also plot the number of bytes returned to the collector (middle) and the number of queries executed per epoch (bottom). We observe that by automatically adjusting σ based on the target c_v, DynATOS+ finds a more optimal value for σ and is able to achieve consistently higher satisfaction compared to DynATOS (a median difference of 18% to 21% for all workload burstinesses) while inducing minimal increased load on collector (a median difference of less than 8 KB per epoch for all workload burstinesses). Note that the number of

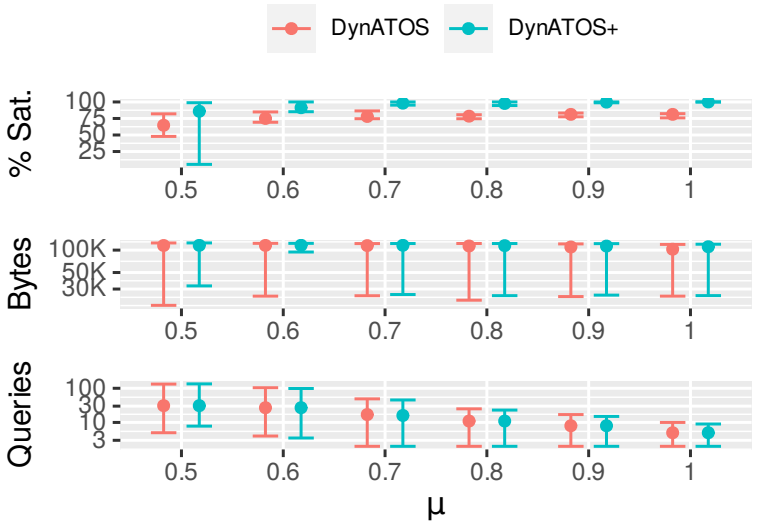


Figure 40. Query satisfaction, bytes sent to collector each epoch, and number of queries each epoch as a function of workload burstiness (smaller μ indicates more bursty workloads) for a single trace showing the improvement of DynATOS over DynATOS.

queries per epoch is larger for burstier workloads because we ignore epochs where no queries were run. Also, in some cases the query workload consists of a single burst of queries over the entire 15 m trace so that DynATOS+ does not have a sufficient number of epochs in which to tune the c_v to σ translation. This causes the lower error bar for DynATOS+ at $\mu = 0.5$ compared to DynATOS which uses a fixed (in this case higher) σ. In a real deployment where DynATOS+ is run for longer periods (*e.g.,* several hours), we expect query satisfaction would converge closer to the median in these plots.

5.1.3.4 Interaction Between Traffic and Query Workload Dynamics.

To understand the interaction between traffic dynamics and query workload dynamics, we run the same query workloads used in Figure 40 over all 28 traces in our sample from MAWILab. Figure 41 summarizes the results by showing the

minimum and maximum median query satisfaction and bytes sent to the collector over all 28 traces for each workload burstiness setting μ. As in Figure 40, quantiles (shown here as different colors) are over the same 10 independently-generated workloads at each μ.

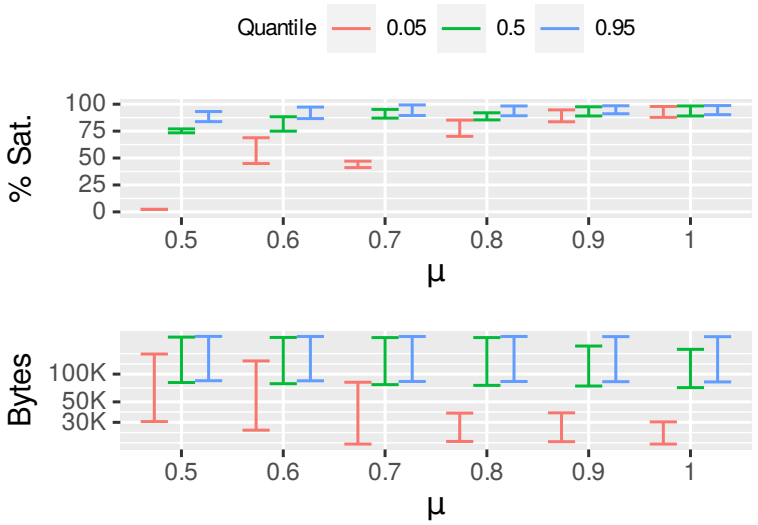


Figure 41. Minimum and maximum over all traces of query satisfaction (for same quantiles as in Figure 40). The differences between different workload burstiness (across x-axis) is much larger compared the the differences among different traces (y-axis ranges) indicating workload burstiness has more impact on DynATOS's performance.

Similar to Figure 40, we observe that burstier workloads achieve lower satisfaction ($\sim$73% for $\mu = 0.5$ compared to $\sim$89% for $\mu = 1.0$). On the other hand, the distance between minimum and maximum satisfaction across all traces (vertical height of the bars in Figure 41) is relatively consistent and small for all query workloads (an absolute difference of median from $\sim$4% to $\sim$13% for all workload burstinesses). The relatively larger range of bytes returned to the collector ($\sim$68% difference in median across all workload burstinesses) demonstrates how DynATOS's sampling method automatically adjusts load on the collector to meet the different number of keys

in different traces. Considering the fact that the per-epoch CV of number of keys considered ranges from ∼0.04 to ∼0.4 across the traces in our sample, these results indicate the methods in DynATOS are more robust to dynamics of traffic composition compared to burstiness of query workloads. We envision developing new methods to improve DynATOS's handling of bursty query workloads as future work.

5.1.3.5 Sensitivity to Epoch and Subepoch Duration. The two most critical parameters in DynATOS are the duration of epochs and the duration of subepochs. Since epoch duration determines the lower bound on how quickly query results can be delivered, we assume network administrators fix epoch duration based on their particular monitoring requirement. The key remaining parameter is then the duration of subepochs or equivalently the number of subepochs per epoch.

To understand the impact of subepoch duration on DynATOS's performance, we run DynATOS over the same trace considered in § 5.1.3.3 and the ten independent query workloads with $\mu = 1$. To expose the impact of query operation multiplexing, we limit the number of queries that can be assigned to run in a single epoch to twice the mean expected number of queries (*e.g.,* because the mean query arrival rate is 1 per second, for 8 s epochs we limit to 16 concurrent queries in each subepoch). Our query workload driver does not attempt to resubmit queries that are rejected when DynATOS is in "fail-safe" mode leading to some fraction of queries going unanswered in some runs.

Figure 42 shows the resulting F1 score (top), bytes sent to the collector (middle), and fraction of queries submitted by the workload that actually receive answers, regardless of their accuracy goals (bottom). Note that due to the minimal progress constraint (C4 in Table 5), we are limited to a minimum of two subepochs per epoch. Overall, F1 score is not significantly impacted by epoch duration indicating that

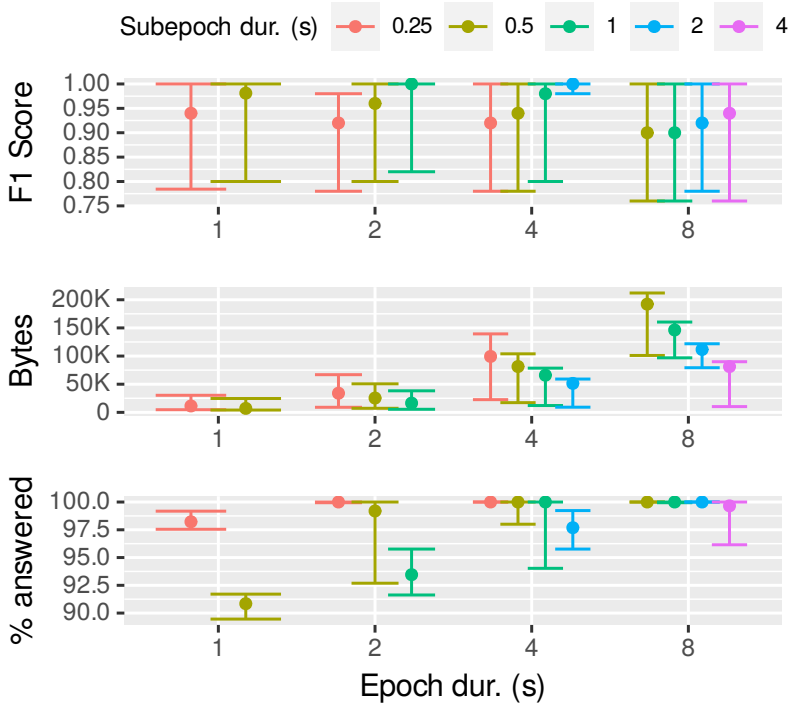


Figure 42. Impact of epoch and subepoch duration on query accuracy, volume of traffic to collector each epoch, and fraction of queries answered.

network administrators can confidently choose the epoch duration best suited for their monitoring requirements without impacting DynATOS's performance.

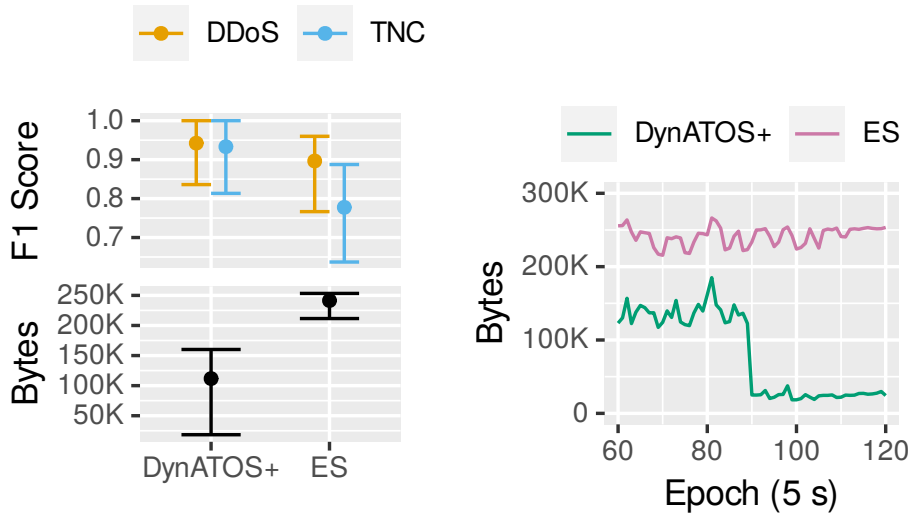
Subepoch duration has a more complex effect on all metrics observed. First, longer subepochs tend to yield slightly higher accuracy (*e.g.,* median F1 scores increases from 90% to 94% for 8 s epochs) for queries that receive an answer. Second, shorter subepochs cause more bytes to be sent to the collector each epoch because intermediate results must be sent after each subepochs (*e.g.,* from 81K up to 192K for 8 s epochs). These two facts may seem to suggest that simply setting subepoch duration as long as possible yields optimal performance. However the bottom plot in Figure 42 exposes the cost of having fewer longer subepochs: due to the limited ability to multiplex query operations, a larger number of queries are rejected due to DynATOS being in "fail-safe" mode. Moreover, fewer queries are answered for

shorter epoch durations (*e.g.,* with two subepochs, a median of ∼90.8% for 1 s epochs compared to ∼99.9% for 8 s epochs) due to the fact that longer epoch durations smooth over bursts of queries that otherwise lead to infeasible scheduling problems at shorter epoch durations. In summary, we find that using between 4 and 8 subepochs exposes a sweet spot between *(i)* higher error and traffic to collector with more than 8 subepochs and *(ii)* reduced possibilities for multiplexing which cause more queries to go unanswered with fewer than 4 subepochs.

5.1.3.6 Comparison with Sketch Methods. We compare the empirical performance of DynATOS to two relevant sketch-based proposals: ElasticSketch [176], which includes provisions for dealing with changes in traffic composition, and Newton [187], which includes provisions for running dynamic query workloads by allow queries to be changed at runtime.

ElasticSketch. Although ElasticSketch can adapt to some types of change in traffic composition (*e.g.,* flow-size distribution), it does not support efficient changing of queries at runtime. To illustrate, we consider a simple query workload which runs the DDoS query for 7.5 minutes, then switches to the TCP new connections query for the next 7.5 minutes. Note that this workload represents a wide range of scenarios where two or more queries with different filter conditions and different keys are required to run in sequence. We run two independent instances of ElasticSketch in parallel (one for each query). Since ElasticSketch does not support runtime reconfiguration on switch hardware, both instances run throughout the workload even though only one query output is used at a time. DynATOS, on the other hand, can simply switch between queries after 7.5 minutes so only one query is run at a time.

We run this scenario over traffic from the May 29 trace in our sample since this trace has relatively dynamic composition (as measured by CV of number of



(a) Summary quantiles.

(b) Load on collector

Figure 43. Performance of DynATOS and ElasticSketch on dynamic query scenario where the network administrator changes between DDoS and TCP New Connections queries. ElasticSketch requires sending >2× more bytes to the collector while achieving lower accuracy compared to DynATOS because it cannot adapt to the change in queries.

keys per epoch). We set $c_v = 0.1$ and adjust the size of ElasticSketch to achieve slightly lower accuracy compared to DynATOS as measured by F1 score. Figure 43a shows the actual F1 score achieved (top) as well as the number of bytes sent to the collector (bottom) over all epochs for the two methods described above. Even though it achieves lower accuracy for both queries, the overheads of maintaining parallel sketches in ElasticSketch require sending over 2× more bytes to the collector compared to DynATOS (∼240KB compared to ∼110KB). As shown in Figure 43b, DynATOS achieves lower load on collector by only sending results for one query at a time. In general, this example illustrates that even for simple dynamic query workloads, the ability to switch between queries at runtime leads to significantly lower overheads.

Newton. Newton [187] develops methods to dynamically change queries on-the-fly (similar to DynATOS), but the Newton dataplane uses fixed-size sketch-based primitives which cannot adapt to changes in traffic composition.[1] To illustrate, we consider a scenario where the network administrator runs the Port Scan query over a trace from our MAWILab sample with pronounced changes in traffic composition (in particular Aug. 12). In particular, this trace has a relatively constant number of keys per epoch (∼18K) until the 135-th epoch when the number of keys spikes up by an order of magnitude (to ∼273K). We set the sketch sizes in Newton to achieve high accuracy on the first part of the trace and choose $c_v = 1.5$ so that DynATOS achieves slightly lower accuracy compared to Newton for the first part of the trace.

Figure 44 shows, for each epoch of the trace (x-axis), the F1 score (top), fraction of bytes sent to collector compared to ground-truth (middle), and total number of keys in the underlying (ground-truth) traffic (bottom). Before the change in composition at epoch 135, Newton achieves high F1 score (median of 1.0) compared to DynATOS (median of 0.8). However, when the number of keys changes, Newton's sketches become full leading to a significant reduction in F1 score (median of 0.24). The middle plot shows that the root cause of this is the relatively smaller number of bytes Newton sends to the collector compared to ground truth. DynATOS, on the other hand, achieves consistent F1 score (median of 0.8) during the increase in number of keys by maintaining a consistent load on collector w.r.t. the total number of ground-truth keys that need to be reported.

5.1.4 Takeaway. In this section we presented DynATOS, an end-to-end telemetry system design that leverages the temporal-structure-aware designs described in § 4.1. We showed how DynATOS achieves better performance compared

[1]In particular, the Port Scan query considered here uses a Bloom filter [37] with a fixed number of bits to approximate the first "distinct" operator.

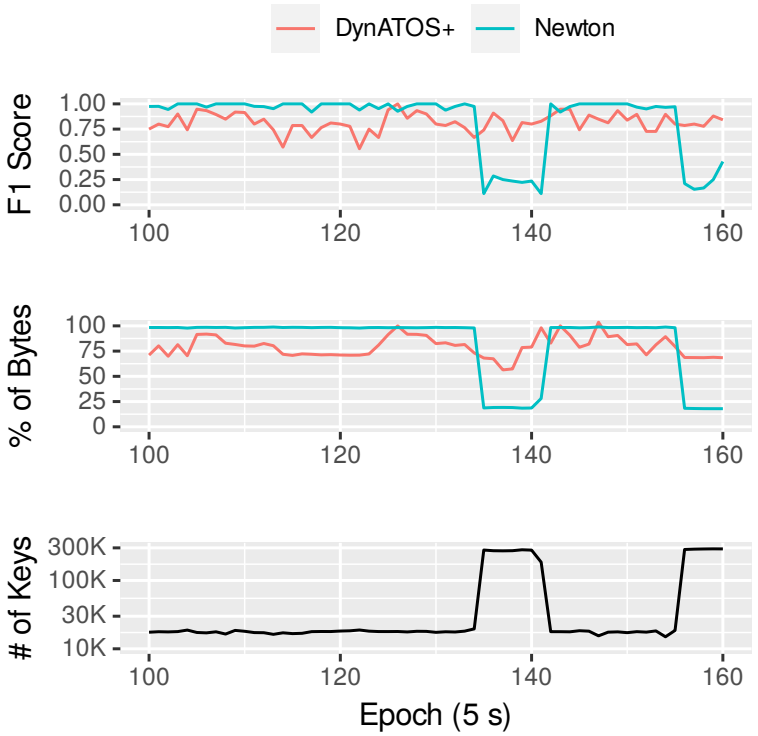


Figure 44. Performance of DynATOS and Newton running the Port Scan query on a excerpt from our sample of MAWILab traces with heavy traffic dynamics. Even though Newton is tuned for high accuracy, when the number of keys in the underlying traffic changes it suffers significant accuracy loss.

to sketch-based alternatives in real-world traffic scenarios where the number of groups changes dramatically. Moreover, we showed how our approximation and scheduling methods can achieve robust accuracy on a wide range of different traffic and query workloads. In doing to, we close the line of investigation stemming from the observed high-variance structure of the number of groups per time unit (§ 3.1) with a practical telemetry systems that demonstrates improved performance over state-of-the-art for queries that compute per-group metrics. We next describe a system that detects particular groups suspected of sending malicious attack traffic.

5.2 Detecting DDoS Attack Traffic with ZAPDOS

This section describes how we leverage the spatial design elements and algorithms introduced in § 4.2 to develop a concrete system to automatically identify the signature of volumetric DDoS attack traffic. In particular, in § 5.2.1 we describe the problem space of volumetric DDoS attack signature detection and the design choices made in ZAPDOS. Then, § 5.2.2 describes several non-trivial engineering techniques used in our implementation of a ZAPDOS prototype for programmable switch hardware and § 5.2.3 describes our evaluation of ZAPDOS on realistic attack scenarios. Finally, § 5.2.4 considers several potential ways a skilled adversary could circumvent the protections offered by ZAPDOS and how minor extensions of our designs could mitigate these circumventions.

5.2.1 ZAPDOS Overview and System Design.

In this section, we first describe our threat model and then provide a brief overview of ZAPDOS, highlighting its key contributions and challenges.

5.2.1.1 Threat Model.

Volumetric Distributed Denial of Service (DDoS) attacks [115] are a persistent threat for Internet-connected devices caused by the inherently open property of the Internet that any device can send traffic to any other device (so long as both devices have public-facing IP addresses). In a volumetric DDoS attack, a skilled adversary uses this property to send a large volume of unwanted "attack" traffic from a large number of sources towards a victim device or network. Intuitively, if the attacker is able to send more traffic than the victim is able to process, benign traffic to the victim will experience increased loss and, in the worst case, the victim will be completely disconnected from the Internet. Volumetric DDoS attacks continue to grow in volume and complexity, achieving multi-terabit traffic volumes [160] and changing attack sources and vectors dynamically to avoid

detection [106, 103]. We formalize the problem of detecting volumetric DDoS attack signatures through considering the attacker's and victim's metrics of success in the following.

Attacker's metrics of success. We assume a rational attacker whose goal during the active-attack phase is to inflict maximum damage (*i.e.,* loss) on the targeted network's benign traffic while minimizing the cost of launching the attack.

If the victim network has no defense, damage is measured in how much benign traffic is lost due to attack-induced congestion. Based on recent studies [163, 87], we assume the attacker uses one of several reflection attack vectors (e.g., DNS, NTP, SSDP) and/or botnets sending flooding attacks (e.g., SYN, ICMP, UDP). To launch these attacks the main cost for the attacker lies in acquiring sufficient attack sources (*e.g.,* bots) to maximize attack volume.

If the victim network deploys signature-based defenses, the notion of damage as well as cost of attack is more complex. First, effective signature-based defenses increase the cost for the attacker since extra effort is required to evade detection. In particular, the attacker can combine several different attack vectors and change attack vectors and attack sources dynamically.[2] Second, signature-based defenses also introduce an additional type of damage in that the reported signatures may falsely include benign sources. The attacker can potentially leverage this by intelligently selecting attack sources to confuse or mislead signature detection. We assume the adversary uses one of three methods to select attack sources: *(i)* based on the actual IP address of the reflector[3] or bot; *(ii)* based on a fixed uniform random sample of

[2]Due to the overheads of precise clock synchronization (*e.g.,* access to GPS receivers) we assume a lower bound on how fast the adversary can coordinate these changes across their bots (*e.g.,* can only change attack parameters once every second).

[3]Note that, although reflection attacks do require bots to spoof source addresses, the adversary is unable to spoof the addresses of reflectors which are observed at the victim network.

the source address space; or *(iii)* based on proximity to benign traffic through the cost-based method. In the later case, we assume the adversary cannot guess exactly which sources will appear in benign traffic, but can pay a higher cost (in terms of effort during the pre-attack phase) to infer attack sources that have longer prefix overlap with benign sources.

Victim's metrics of success. The goal of the victim is to prevent as much attack-induced damage as possible, from either flooding loss or false-positive signatures. We assume the victim is able to detect when a volumetric DDoS attack occurs but does not have any additional information about the attack (*i.e.,* the victim has no prior knowledge of the attack vector or the attack sources). Anecdotal evidence suggests that for small to medium scale edge networks, volumetric DDoS attacks often cause spikes in packet and bit rates which can be detected using simple counters (*e.g.,* on the border switch in Figure 1). A variety of other volumetric DDoS occurrence metrics have been proposed and implemented on switch hardware [70, 52] and could be integrated with ZAPDOS. Finally, we assume the victim has the ability to deploy traffic monitoring for signature detection and the ability to mitigate detected attack traffic (*e.g.,* rate-limit, re-route through a scrubbing service).

Note that the assumptions made in our threat model about the type of attacks and methods for detecting attack occurrences are consistent with others (*e.g.,* [103, 52, 24]).

5.2.1.2 Overview of ZAPDOS. As shown in Figure 45, ZAPDOS is a closed-loop hybrid hardware-software approach to detecting prefix-level volumetric DDoS attack signatures. Operating during the active-attack phase, ZAPDOS uses a fixed set of *prefix monitoring slots* implemented in programmable switch hardware, control software, and a set of operator-defined mitigation policies. ZAPDOS uses the prefix monitoring slots to collect features for a fixed number of prefixes at a

time, reacts to the results by updating which prefixes to monitor, and reports attack prefixes as soon as they are known to be separate from benign prefixes with sufficient confidence.

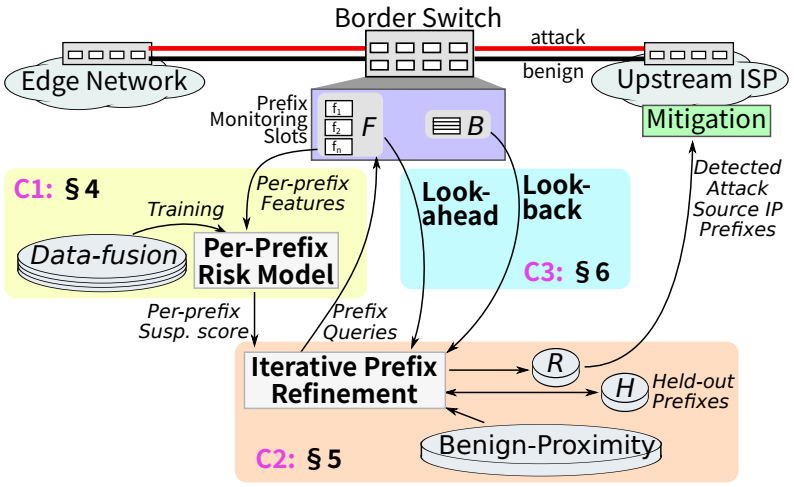


Figure 45. Overview of ZAPDOS.

The key contributions in ZAPDOS are concrete and novel solutions to the following challenges.

C1: How to determine if aggregate prefix-level traffic contains an attack source? The per-source or per-flow feature vectors used in prior ML-based approaches [54, 32] supply detailed signals directly correlated to the traffic entities to be classified. The per-prefix feature vectors considered in ZAPDOS, on the other hand, may contain signals from a variety of attack and benign sources and ZAPDOS's model must be able to "see through" these ambiguous prefixes in order to effectively refine and detect an accurate attack signature.

ZAPDOS trains a classification model over prefix-level features from a large number of scenarios with realistic packet-level attack traffic as well as attack and benign source address distributions, as described in Appendix C. In order to increase the model's ability to detect the presence of attack traffic in ambiguous prefixes, we

develop a prefix length weighting method to explicitly bias the model's FNR vs. FPR tradeoff at different prefix lengths.

C2: How to use fixed switch resources to monitor a variable number of variable-length prefixes? The simplest iterative refinement approach simply zooms in or expands any prefixes that appear to contain any amount of volumetric DDoS attack traffic; however, this approach quickly leads to a large and infeasible number of prefixes to monitor. Since the total number and length of prefixes involved in an attack signature is unknown in the pre-attack phase, prior work which partitions the address space into a fixed number of prefixes (*e.g.,* DREAM [120]) is also insufficient.

ZAPDOS leverages the refinement algorithm described in § 4.2.1 to carefully decide which prefixes are worth monitoring and when particular prefixes can be reported as sourcing attack-only traffic with high confidence. The core idea of this algorithm is to partition prefixes as they are zoomed in on between a fixed-sized set of high priority prefixes to assign to monitoring slots and a variable-size set of less suspicious "held-out" prefixes. A recent snapshot of the structure of benign sources from the pre-attack phase is used to determine when a suspicious prefix has been zoomed in on enough to minimize false positives.

C3: How to ensure the refinement process remains robust when the adversary changes attack sources and vectors? A key limitation of iterative refinement approaches, including the base-line approach describe above, is that they assume the attack signature does not change rapidly. Modern DDoS attacks, on the other hand, may rapidly (e.g., every 30 s) change attack sources in order to confound such approaches.

ZAPDOS leverages the two improved refinement techniques discussed in § 4.2.2 to allow the refinement process to quickly adapt to changes in the underlying attack

traffic. First, ZAPDOS uses "look-ahead" to detect which child prefixes are active (whether with attack traffic or benign traffic) *before* allocating precious monitoring resources in the next epoch. Second, ZAPDOS uses "look-back" to keep a low-overhead approximate record of which non-monitored prefixes were active to guide resource allocation in the next epoch when the attack sources change.

5.2.2 ZAPDOS Prototype. We prototype ZAPDOS using a Tofino-enabled Wedge 100BF-32QS switch[4]. The switch ASIC runs a P4 [39] program that computes per-prefix features in hardware registers (Table C.3), maintains look-ahead and look-back components (§ 4.2.2), and classifies packets associated with reported attack prefixes in R. Table 7 shows that ZAPDOS has a relatively small footprint compared to the available hardware resources for key resource types like SRAM and TCAM, comparable to the footprint of Jaqen[5].

Crossbar	**SRAM**	**TCAM**	**VLIW**	**Hash Bits**	**ALU**	**Gateways**
6.70 %	7.29%	11.11%	9.38%	17.11%	26.04%	15.62%

Table 7. Switch hardware resources used by the ZAPDOS data plane as percentage of total available on Tofino.

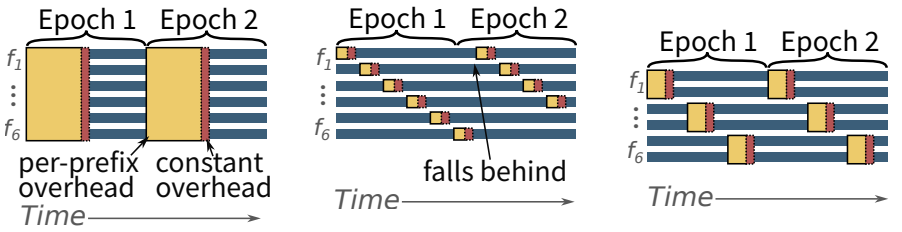
The switch CPU runs the ZAPDOS control plane, a Haskell [9] program which evaluates the per-prefix classification model (Appendix C) and refinement algorithm (§ 4.2.1), interacting with the ASIC through the `bfruntime` gRPC interface and ASIC's CPU port. A key challenge in prototyping ZAPDOS in hardware is optimizing the communication between ASIC registers and the switch's CPU to minimize the update time overhead. In addition to carefully multi-threading the ZAPDOS control

[4]`https://www.edge-core.com/productsInfo.php?cls=1&cls2=5&cls3=181&id=770`

[5]Note that Jaqen only reports hardware resource usage of their detection module, but their mitigation modules, which actually generate source-level attack signatures, require unknown additional resources.

plane, we leverage two other techniques to optimize this communication: *(i)* batch-round-robin epochs and *(ii)* packet ferries.

5.2.2.1 Batch-Round-Robin Epochs. Naïvely reading the features of all prefix monitoring slots in one shot at the end of each epoch (as shown in Figure 46a) requires a significant amount of time. For example, writing a new set of 1500 prefixes to monitor in hardware takes ∼0.5 s, implying traffic would only be monitored half of the time with 1-second epochs. On the other hand, incremental updates that cycle through prefixes in round-robin order (as shown in Figure 46b) cause the iterative refinement process to fall behind due to the constant overhead associated with each RPC. For example, an RPC updating a single prefix takes ∼5 ms, implying one cycle through all 1500 prefixes would take ∼7.5 s.



(a) Sync. Epochs (b) Round-Robin (c) Batch-Round-Robin

Figure 46. Illustration of different possible approaches to updating prefix monitoring slots in ZAPDOS.

In ZAPDOS, we develop an approach called *batch-round-robin* which updates batches of prefixes in a single RPC and cycles through batches in round-robin order as shown in Figure 46c. This combines the benefits of synchronous epochs and round-robin updates because the constant overhead of each RPC is amortized over all prefixes in a batch and each batch is much faster to update compared to the entire set of monitoring slots. Updating a batch of 100 prefixes in our prototype takes ∼25

ms so the 15 batches required to update all 1500 prefixes takes 375 ms implying batch-round-robin can easily keep up with 1-second epochs.

5.2.2.2 Packet Ferries for Feature Collection. To implement the modeling and iterative refinement methods, the ZAPDOS control plane needs to read features of all 1500 prefix monitoring slots back to the CPU each epoch, but a naïve batch-read RPC request takes ∼1.3 s. Instead, ZAPDOS implements a technique called *packet ferries*, inspired by in-band network telemetry [89, 35] and further developed in [146]. As shown in Figure 47, packet ferries send specially-marked request packets through the ZAPDOS data plane using the switch CPU's backplane port which bypasses gRPC and driver layers allowing ZAPDOS to read all 1500 prefixes in ∼50 ms.

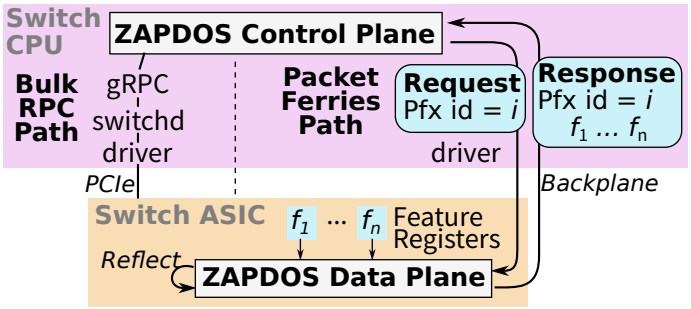


Figure 47. Packet ferries enable fast reads by bypassing the gRPC and driver layers.

5.2.2.3 Asynchronous updates to R and H. The policies for how to select the set of prefixes to monitor F described in § 4.2.1 and § 4.2.2 require zooming-in on and partitioning a monotonically increasing set of prefixes based on classification results produces by the model described in Appendix C. At first glance, batch-round-robin can be applied to implement these policies by simply passing each batch through the algorithm independently and concatenating the results. However, several more subtle challenges arise: *(i)* the number of newly zoomed-in-on children

exceeds the batch size; *(ii)* the held-out prefixes in H need to be looked up in the look-back Bloom filter B; and *(iii)* newly reported prefixes in R need to be sent to network administrators.

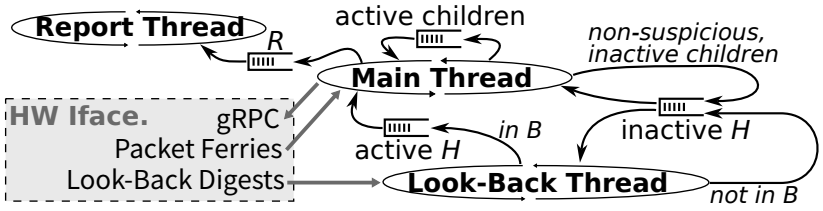


Figure 48. Distinct threads and inter-thread communication patterns in the ZAPDOS control plane.

Our approach, summarized in Figure 48, splits the refinement process into several independent threads that run in parallel and communicate via asynchronous multi-reader, multi-writer queues.[6]

Main thread. The main thread implements batch-round-robin over a fixed set of hardware prefix monitoring slots. After reading features from hardware using packet ferries, applying the model, zooming-in on suspicious prefixes, and partitioning the results based on look-ahead, this thread installs the first `batchSize` prefixes to monitor in hardware using gRPC and pushes the remaining prefixes to several queues. In cases where fewer than `batchSize` prefixes were selected as suspicious, this thread pulls new prefixes to monitor from the queues based on which queues are non-empty. Finally, when prefixes are determined to be ready for inclusion in R (§ 4.2.1.3), they are pushed to a dedicate report queue.

Look-back thread. A naive approach to implementing look-back in ZAPDOS would be to wait until fewer than `batchSize` prefixes are available from other sources, then to query the hardware Bloom filters for prefixes in H until an active prefix is

[6]In particular, we use unagi-chan [19].

discovered. However, given the non-negligible latency of directly reading ASIC tables and the need to have multiple Bloom filters in hardware (one for each possible prefix length) this approach is highly inefficient. In our ZAPDOS prototype, we implement a single /32 Bloom filter in the ASIC and use *digests*[7] to report each time a new /32 source is observed. The look-back thread receives these digests and uses them to update a *look-back table* tracking which prefixes have been observed each epoch. At the end of each epoch (*e.g.,* every 1 second), the look-back thread pulls all held-out prefixes from H and partitions based on their membership in the look-back table. Prefixes from H which were active are pushed to an *active* queue while prefixes that were inactive are pushed back to H. In order to implement look-back as described in § 4.2.2.2, the main batch-round-robin thread simply empties the active queue before pulling from the inactive prefixes in H.

Report thread. A key advantage of ZAPDOS is that network administrators can respond to attack prefixes reported in R in a wide variety of different ways. However, realizing these responses may induce additional latency (*e.g.,* additional RPC or REST requests) which could impact progress towards iterative refinement. To address this, we implement a dedicated attack report queue and attack report thread which executes custom attack responses independently from iterative refinement. In our current prototype, the attack report thread simply writes each reported prefix to a file along with metadata (*e.g.,* timestamps) for scoring against ground-truth after each trial run.

5.2.3 Evaluation.

We build a large dataset of realistic attack traces using the methodology described in § C.1 and use this dataset to evaluate ZAPDOS. We show that ZAPDOS

[7]Digests are short messages pushed from the ASIC to the switch CPU through the gRPC interface and commonly supported in P4 platforms like Tofino.

- quickly detects attack signatures with high accuracy in our hardware prototype implementation and we provide a breakdown of the latency overhead (§ 5.2.3.2),
- accurately detects signatures of modern multiple-vector and dynamic attacks with a single model (§ 5.2.3.3),
- maintains low error rates for changes to attack parameters (*e.g.,* numbers of sources), system parameters (*e.g.,* `prefixesPerEpoch`), and loss scenarios (*e.g.,* flooded border links) (§ 5.2.3.4), and
- is robust against attackers who can spoof attack traffic to come from the same prefixes as benign traffic maintaining orders of magnitude lower error rates compared to prior approaches when given comparable traffic monitoring resources (§ 5.2.3.5).

5.2.3.1 Setup.

We next describe our experimental setup.

Success metrics. The goal of ZAPDOS's attack signatures are to effectively reduce the amount of damage caused by an attack regardless of how much effort the adversary spends on the attack as described in § 5.2.1.1. To measure potential damage, we measure the accuracy of given attack signatures w.r.t. ground-truth in terms of per-byte *false positive* (FPR) and *false negative* (FNR) rates. Smaller FPR and FNR indicate more accurate attack signatures and in turn less damage for the target network. To measure how fast an attack is detected, we use the *detection time* metric that we define to be the time difference between when an attack starts and when ZAPDOS reports more than a certain fraction of the attack prefixes. Note that we classify each packet as a false-positive or false-negative based on the current value of R when that packet arrived and compute aggregate FPR and FNR over the entire attack scenario. As a result, these metrics also reflect how quickly ZAPDOS was able to refine accurate attack signatures (*e.g.,* if ZAPDOS takes longer to detect a

particular attack prefix, it will make a larger contribution to FNR). The shown error bars mark the 5^{th} percentile, the median, and the 95^{th} percentile, respectively over independent trials.

Other DDoS defense approaches. We compare ZAPDOS with two state-of-the-art switch-based approaches to volumetric DDoS defense in edge networks: *(i)* Euclid [52] uses a sketch-based method to detect attack sources by estimating their contributions towards the increase in the entropy associated with the attack, and *(ii)* Jaqen [103] is a library of sketch-based detection and mitigation primitives that can be deployed on switch hardware with a CPU-based controller. We use an exact heavy-hitter computation to represent the best-case scenario for Jaqen, which originally used universal sketching to estimate heavy hitters. We do not compare with approaches like Poseidon [181] as it focuses on local solutions with no clear separation of detection and mitigation. We also do not compare with ML approaches like LUCID [54] since their feature-gathering overheads are infeasible for edge networks.

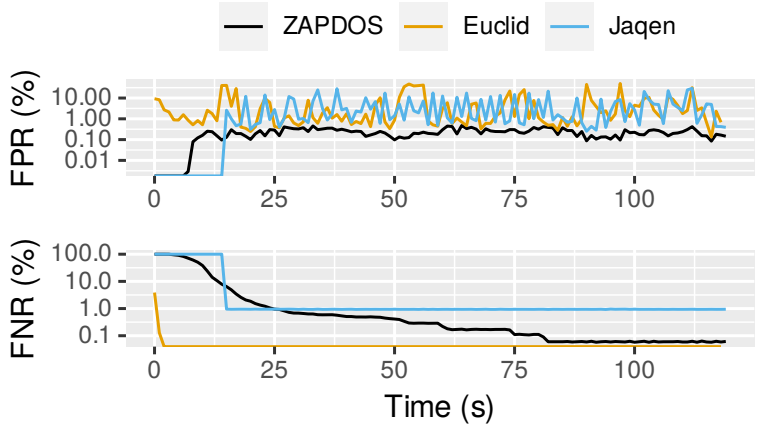
Default parameters. As discussed in § 4.2.1 and § 4.2.2, the two main parameters controlling efficiency of iterative refinement in ZAPDOS are `prefixesPerEpoch`, and `zoomInBits`. Unless otherwise noted, we set `prefixesPerEpoch` to 1500 and `zoomInBits` to 4 since these values yielded acceptably low error rates in initial experiments and acceptably low overheads in our hardware prototype. We use a Bloom filter with 2^{20} bits and a single hash function in our prototype for look-back. We set a default epoch duration of one second. The pre-attack phase lasts for 120 s to allow ZAPDOS to compute the benign traffic profile described in § 4.2.1.3 and we set the benign proximity threshold to 0. Unless otherwise noted, the active-attack phase lasts for 120 s with an aggregate attack rate of 1 Gbps (enough to flood an access link of a small or medium-sized campus network) using 50k attack sources from

Mirai data [21] (distinct from the training sources) comparable in size to real-world volumetric DDoS attacks [141].

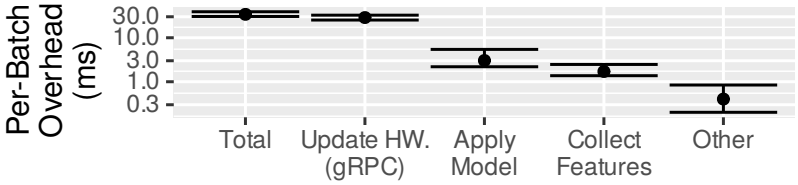
5.2.3.2 ZAPDOS Prototype Performance. To evaluate our prototype, we replay an independent UDP-flood attack trace from the test partition of our dataset containing a mix of attack and benign traffic.[8] The prototype is switched from collecting the benign traffic profile (§ 4.2.1.3) to actively zooming in on attack prefixes when the trace enters the active-attack phase at 120 s. We collect the list of reported attack prefixes in a file which also includes a timestamp of when each prefix was reported, then compute per-byte FPR and FNR in each 1 s time window based on the timing of these reports offset relative to the original trace.

Figure 49a shows the evolution of FPR and FNR in our prototype over the duration of the attack in comparison with simulated implementations of Euclid and Jaqen on the same trace with similar resource budgets. In ZAPDOS, FNR quickly drops below 1% after 25 s (going down to ∼0.05% by 120 s) as more attack prefixes are correctly reported. FPR remains low and stable around 0.2%. In comparison, Jaqen deploys mitigation after 15 s due to overheads of installing the particular mitigation module for UDP flood attacks, then achieves higher FPR (∼4%) and FNR (∼1%). Euclid's pure data plane approach begins mitigating attack traffic within 1 s and identifies nearly all attack sources, but also produces erratic FPR (up to 50%) indicating significant impact on benign traffic. This result demonstrates that ZAPDOS not only effectively detects volumetric DDoS attack signatures in switch hardware, but also consistently achieves lower error rates compared to best-case simulation performance of Jaqen and Euclid.

[8] In particular, we use cost-based attack sources with attack cost $\ell = 16$ combined with MAWILab 2019-02-05 benign traffic.



(a) Timeseries of ZAPDOS prototype accuracy compared with simulation results of Euclid and Jaqen on the same trace with comparable resource budgets.



(b) Characterization of the batch-round-robin update time overhead with 100 prefixes per batch.

Figure 49. Performance of our ZAPDOS prototype on a realistic attack scenario.

We also measured the median absolute difference in accuracy between our ZAPDOS prototype and ZAPDOS simulator over all epochs considered in Figure 49a to be ∼4.1e-04% for FPR and ∼0.17% for FNR. We attribute the slightly larger gap in FNR to inaccuracies outside of ZAPDOS in the tool used to replay attack and benign attack traffic.[9] Given this close correspondence between prototype and simulator, in the rest of this section we show results from our simulation since it enables higher-confidence accuracy computation and evaluation of multiple traces in parallel.

[9]In particular we use `tcpreplay` which appears to fall behind realtime during the active-attack phase leading to delayed timestamps from the prototype which inflate FNR.

Finally, we measure the latency overhead of our batch-round-robin method (§ 5.2.2.1) with 100 prefixes per batch. Figure 49b shows the median total per-batch latency overhead over all epochs of the attack scenario as well as break down across three main operations. Latency overhead is clearly dominated by hardware update time which could be optimized using, for example, DMA rather than gRPC.

5.2.3.3 Performance Against Modern Attacks. **Simple attacks: Single-vector and static.** Modern volumetric DDoS attacks leverage diverse vectors to generate large volumes (*i.e.,* data rates) of attack traffic toward their victims. To demonstrate ZAPDOS's ability to identify attack sources regardless of the attack vector used, we generate an independent single-attack scenario for each of the attack vectors described in Table C.2. For each attack we draw a sample of 50k distinct attack sources from the Mirai [21] dataset, the Booters [141] dataset, or "spoofed" sources from a uniform random distribution (Rnd.). Due to iterative refinement in ZAPDOS, we report in Figure 50 FPR and FNR of the attack signature detected by ZAPDOS for each attack vector at three different points in time after the start of the attack. For all attacks, FNR decreases down to ∼0.1% after 60 s as ZAPDOS reports more attack prefixes. FPR, on the other hand, does not follow a clear trend beyond stabilizing between 0.1% and 0.3% for all attacks. We attribute this to the inherent burstiness of benign traffic which causes burstiness in the FPR time series (*e.g.,* see Figure 49a). Practically, this result demonstrates that the single trained model in ZAPDOS is able to generate highly-accurate attack signatures for a wide range of modern volumetric DDoS vectors within a couple minutes of the onset of that attack.
Complex attacks: Multi-vector and dynamic. In addition to using diverse attack vectors individually, modern DDoS attacks are also known to combine multiple concurrent attack vectors and change them over time [103, 181, 86, 174].

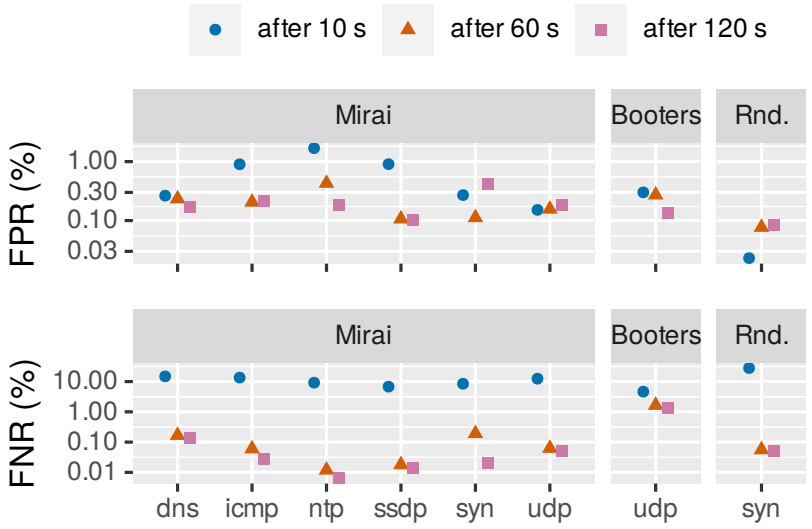


Figure 50. Performance of ZAPDOS on different single-vector, static attacks with 50k distinct sources.

To demonstrate ZAPDOS's ability to handle such complex attacks, we generate multi-vector dynamic attack scenarios by selecting nine distinct attack-vectors from Table C.2 and attack source sets from the Mirai dataset (using the discrete set sizes described in § C.1). During the attack scenarios, at regular intervals the attacker randomly selects three attack vector, attack source set pairs from these nine and combines the traffic of all three against the victim. We generate several such scenarios with different intervals between attack changes, in particular 1 s (fast), 5 s (medium), and 10 s (slow). Figure 51 shows the evolution of attack signatures generated by ZAPDOS during the first 120 s of each scenario. ZAPDOS is relatively slower to refine attack signatures compared to previous experiments because these attacks contain roughly three times the number of distinct sources. However, we note that the rate of signature refinement remains the same regardless of how fast the attack changes between attack sets. This illustrates the effectiveness of both "look-ahead"

and "look-back" methods (§ 4.2.2) to quickly focus limited monitoring resource on target prefixes and to switch prefixes when the underlying attack changes.

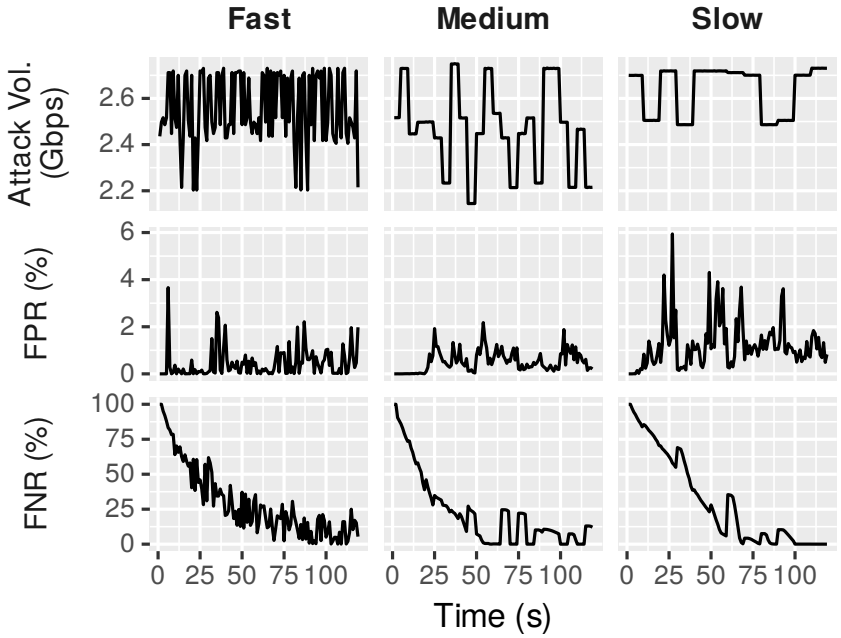
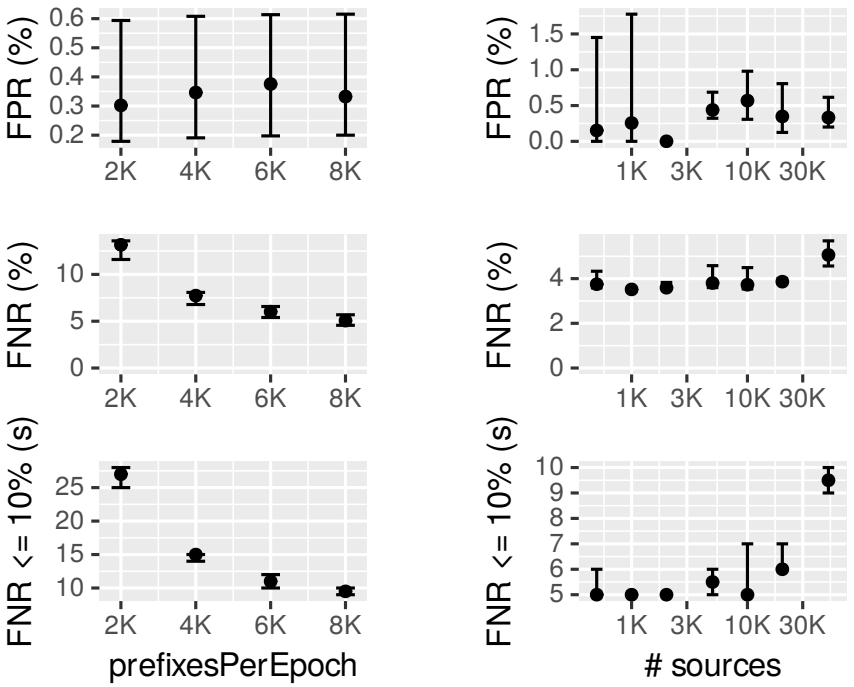


Figure 51. Performance of ZAPDOS in refining signatures of multi-vector, dynamic attacks with varying numbers of sources.

5.2.3.4 Sensitivity Analysis. Since ZAPDOS uses a limited amount of switch hardware memory to monitor attack prefixes, we evaluate how changes to the amount of memory used and to the number of distinct attack sources impact performance. We repeat these evaluations over all 6 attack vectors from Table C.2 and show the minimum, median, and maximum values in Figure 52.

Impact of resource constraints. In Figure 52a, we vary `prefixesPerEpoch` while keeping the number of attack sources constant at 50 k. In all cases, ZAPDOS achieves low FPR (0.3% to 0.4%) independent of `prefixesPerEpoch`. On the other hand, FNR as well as time until FNR is less than 10% decreases as `prefixesPerEpoch` increases since monitoring more prefixes each epoch enables faster progress towards identifying all attack sources.

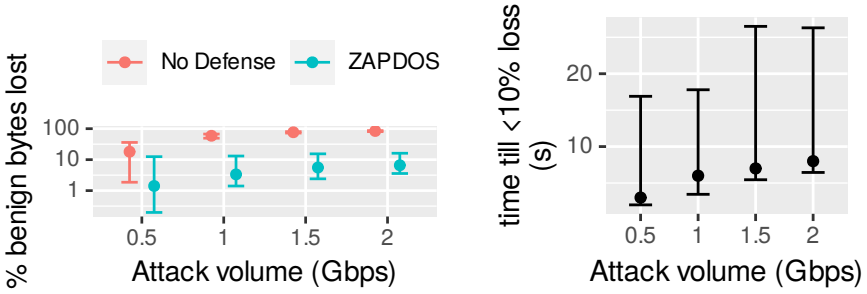


(a) Varying resources. (b) Varying attack parameter.

Figure 52. Performance of ZAPDOS while varying `prefixesPerEpoch` (left) and the number of distinct attack sources (right).

Impact of attack sources. In Figure 52b, we vary the number of distinct sources in attack traffic while keeping `prefixesPerEpoch` at 8 k. ZAPDOS achieves consistently low error rates ($\sim$ 0.5% FPR and $\sim$3.5% FNR) for a wide range of number of sources (500 through 10 k). However, as the number of sources increases, both total FNR and the time until per-epoch FNR drops below 10% begin increasing. We recall that the total attack traffic rate is held constant across these scenarios so that with more sources, the per-source attack rate is significantly reduced. As reflected in Figure 52b, this decrease in per-source attack rate makes the detection problem harder by reducing the relative strength of the attack signals compared to benign traffic hence increasing ZAPDOS's latency and FNR.

Handling Flooded Border Links A volumetric DDoS attack against a small enterprise or campus network at the edge of the Internet can flood the network's main border link, rendering in-network mitigation methods (*e.g.,* Jaqen, Euclid) ineffective and degrading the accuracy of traffic monitoring performed at the edge network. To demonstrate the utility of ZAPDOS's attack signatures in this scenario, we simulate a 1 Gbps border link inserted in front of where ZAPDOS performs its traffic monitoring computations (using a simulated 100 KB FIFO queue). We generate DNS reflection attacks using random samples of 10k sources[10] from the Booters [141] dataset and varying the per-source attack data rate so that the overall attack rate varies from 500 to 2000 Mbps. We simulate upstream mitigation by dropping all packets matching ZAPDOS's reported attack signature.



(a) Total bytes of benign traffic lost with and without ZAPDOS.

(b) Time till benign bytes lost drops below 10%.

Figure 53. Impact of DNS refl. attack volume on defense performance given a 1 Gbps upstream bottleneck link. ZAPDOS reduces benign traffic loss to less than 10% in less than 10 s for the strongest attack (2 Gbps).

Figure 53 shows the performance of ZAPDOS over 10 independent attacks at each attack volume in terms of total damage (volume of benign traffic dropped by the flooded link) and the time it takes before ZAPDOS reduces per-epoch damage to below 10%. We observe that with no defense, the volume of dropped benign

[10]Note that we use a smaller number of sources compared to, *e.g.,* Figure 50 due to the smaller number of sources available in the Booters dataset.

traffic increases consistently with increasing attack volume. On the other hand, Figure 53a shows that signature-based upstream mitigation enabled by ZAPDOS reduces flooding-induced loss to below 10% (median over 10 attacks), even for the largest attack volume (which is 2× the border link's throughput and incurs ∼85% loss without defense). For all attack volumes, ZAPDOS reduces benign loss more than 10×.

Figure 53b shows that larger attacks require longer refinement time before the attack signature is specific enough to reduce loss to below 10%. We observe a similar trend for other percentages of benign traffic loss. This is due to the fact that even though ZAPDOS refines signatures at a similar rate for all attack volumes, in larger attacks, individual attack sources send larger volumes of attack traffic so that a larger number of sources must be blocked before the overall reduction in attack traffic (and hence damage to benign traffic) reaches the same level as for smaller attacks. Nonetheless, even for the 2 Gbps attack, ZAPDOS reduces loss to below 10% in less than 10 s on average.

5.2.3.5 Impact of Proximity of Attack Sources. A fundamental concern with prefix-level attack signatures as detected in ZAPDOS compared with source-level signatures as detected in Euclid and Jaqen is that a resourceful adversary could spoof attack sources to fall into known benign prefixes thereby triggering high false-positive rates.[11] We evaluate this concern by generating 10 independent UDP flooding attack scenarios. Each scenario uses cost-based attack sources as described in § C.1 at four different settings of cost parameter ℓ and distinct benign traffic from MAWILab. Recall that higher-cost attack sources are chosen to share longer prefixes

[11]Note that since Euclid and many of the mitigation primitives in Jaqen are still source-level, cases where attack and benign traffic comes from the same source, *e.g.,* due to NAT, are a common problem across all these methods.

with benign traffic (*e.g.,* for cost $\ell = 24$ all attack sources share a /24 prefix with at least one benign source). We use these scenarios to empirically quantify the impact of cases where attack sources are intentionally crafted to induce high FPR and to compare this impact with performance of Euclid and Jaqen on the same traces using the same methodology as in § 5.2.3.2 above.

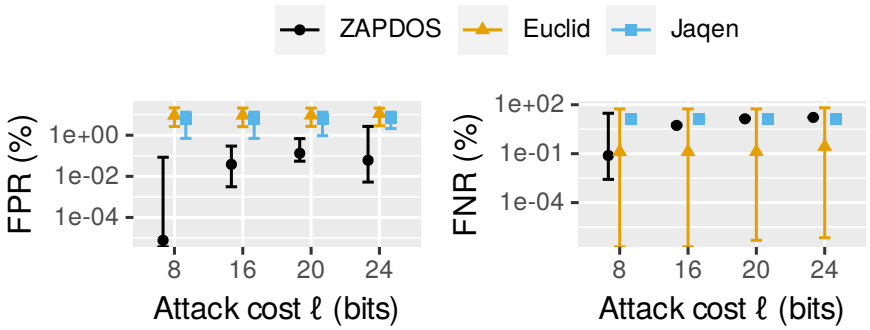


Figure 54. Aggregate performance comparison over 10 independent attacks. Higher attack cost ℓ indicates longer prefix overlap between attack and benign sources.

Figure 54 shows FPR and FNR over all 10 attacks at each setting of cost parameter ℓ. Performance of ZAPDOS does depends on attack cost, with cheaper attacks yielding lower error rates (*e.g.,* median FPR of $\sim 8 \cdot 10^{-6}\%$ at $\ell = 8$ bits compared to 0.06% at $\ell = 24$ bits). However, we note that this trend is concave-down and there is no apparent breakdown of ZAPDOS's prefix-level signatures even up to $\ell = 24$. Although the FPR of Euclid and Jaqen are not impacted by attack cost, they are *orders of magnitude* higher (∼9% and ∼6% respectively) compared to ZAPDOS's prefix-level signatures.

Interestingly, we observe that attack cost also has an impact on ZAPDOS's FNR which ranges from ∼0.07% to ∼17% as ℓ increases. This is a result of longer detection time required by higher-cost attacks as discussed below. In comparison, Jaqen achieves relatively constant FNR around 13% and Euclid's FNR is highly

dependent on characteristics (*e.g.,* entropy) of each trace though uncorrelated with attack cost.

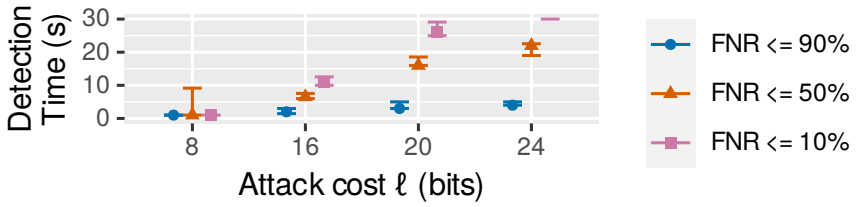


Figure 55. Detection time of ZAPDOS's attack signature coverage.

Intuitively, due to ZAPDOS's benign-proximity method for deciding the length of reported attack prefixes (see § 4.2.1.3), when attack sources are closer to benign sources, ZAPDOS must zoom-in to longer prefix lengths requiring more iterations. Figure 55 quantifies this effect by showing the time between the beginning of the attack and when ZAPDOS's FNR line falls below 10%, 50%, and 90%—in other words, the *detection time* from when the attack starts until when ZAPDOS's attack signature matches 10%, 50%, and 90% of all attack traffic each time window. For the lowest cost attacks ZAPDOS detects 90% of attack traffic within the first 1 s epoch. However, since for higher-cost attacks, attack and benign sources share longer prefixes, ZAPDOS takes longer (up to 24 s in the worst case) to produce sufficiently refined attack signatures. The key takeaway is that if the attacker expends extra effort to place attack sources closer to benign sources, ZAPDOS does not falsely block benign sources, but drills down deeper into the prefix tree to maintain low FPR at the cost of increased signature detection time.

5.2.4 Adversarial Considerations. In addition to attackers who intelligently spoof source addresses to fall within benign prefixes as considered in § 5.2.3.5, several other aspects of ZAPDOS could potentially result in vulnerabilities

which attackers could exploit to thwart detection. We raise several possibilities and note that the severity of each depends on details of a production deployment of ZAPDOS.

Securing communication in ZAPDOS. Since ZAPDOS uses the switch data plane to collect feature results, an attacker could potentially spoof these request or response packets and jeopardize the accuracy of ZAPDOS's features. ZAPDOS's data plane could be modified to include distinct signatures on all result packets (*e.g.,* adding a secret written only through the PCIe channel) allowing the ZAPDOS control plane to verify all received feature results.

Timing-based attacks. A skilled attacker could use knowledge about the epoch duration used in ZAPDOS to craft attacks that inhibit iterative refinement (*e.g.,* by changing attack sources faster than ZAPDOS's epoch duration). In addition to keeping actual parameters used in ZAPDOS private, production deployments of ZAPDOS could also dynamically adjust epoch duration following cryptographically-secure random patterns to mitigate the effectiveness of such attacks.

Model-based attacks. Attackers with detailed knowledge of the data used to train the inference model used in ZAPDOS could discover ways to trick the model into making false positive or false negative decisions. In addition to keeping training data used in production private, the models used in ZAPDOS could be periodically re-trained based on the most recent features of benign traffic observed on the victim's network to reduce the opportunity for such model-based attacks.

5.2.5 Takeaway. In this section we presented ZAPDOS, an end-to-end system for automatically detecting the prefix-level signature of modern volumetric DDoS attack traffic that leverages the spatial-structure-aware designs described in § 4.2. We showed how ZAPDOS achieves better performance compared to state-of-

the-art programmable switch hardware attack detection and mitigation proposals and how ZAPDOS effectively handles a wide range of different types of modern attacks. In doing so, we close the line of investigation stemming from the observed multifractal structure of real-world IP addresses (§ 3.2) with a practical traffic monitoring system, thereby demonstrating the real-world significance of the connections between research areas proposed in Chapter I. We next consider several interesting open problems in each of the areas of research touched on in this dissertation.

CHAPTER VI

CONCLUSION & FUTURE OUTLOOK

This dissertation developed contributions to state-of-the-art in network traffic monitoring by leveraging the connections between three interrelated research areas: characterizing the structure of network traffic, designing algorithms for traffic monitoring, and solving real-world traffic monitoring tasks. In Chapter II we discussed prior work related to network telemetry system design and showed how key open problems relate to insufficient consideration of connection between related research areas. In Chapter III we developed particular characterizations of temporal and spatial structure of network traffic relevant to telemetry system design. In Chapter IV we demonstrated how these characterizations of traffic structure inspire and motivate new approaches to designing traffic monitoring algorithms for two particular types of traffic queries. Finally, in Chapter V we showed how our design approaches can be integrated in real-world telemetry systems and how the resulting systems achieve higher performance in terms of accuracy and efficiency compared to prior approaches.

Our work demonstrates how strengthening connections between areas can improve state-of-the-art for particular types of traffic structure, queries, and system deployment settings. However, the particular contributions we made in and between each area are only points in the larger space exposed by the proposed paradigm of seeking solutions through strengthening connections between areas. We believe this paradigm itself lends value to telemetry system research by showing the way forward for future work in a wider range of research problems. To illustrate, the following considers several example future directions in each research area considered.

6.1 Future Characterizations of Traffic Structure

As discussed in § 2.1.1, the primary focus of efforts to characterize the temporal structure of network traffic focus on data rates like packet count per time unit. Though § 3.1 provides an initial characterization of the time-domain structure of the number of distinct groups per time unit, more investigation is necessary to fully describe the observed patterns of non-fractal periods with rare, but significant increases in the numbers of certain types of groups (*e.g.,* the spike in number of source addresses described in Figure 6). Additionally, longer-term views of these time series are known to exhibit various forms of seasonality (*e.g.,* daily patterns based on business ours, weekly patterns based on workdays, etc.). Fully characterizing the time-domain behavior of the number of distinct groups observed will likely require a novel combination of existing statistical techniques to account for this diverse set of factors.

Though § 3.2 updates a critically lacking aspect of our understanding of the spatial structure of observed addresses, it focuses more on the processes which motivate the cascade construction as an appropriate multifractal model and the connections to telemetry system design. Further exploration of different techniques of statistical analysis, such as discrete wavelet leaders [84], which have been shown to provide higher-confidence characterizations (*e.g.,* including empirically derived confidence intervals [169, 170]) are left for future work. Similarly, more formal and wide-scale analysis of the same types of address assignment policy data (*e.g.,* bulk whois records) to further solidify the cascade construction and foundational multifractal scaling of observed addresses are necessary.

Finally, although the independent temporal and spatial traffic structure characterizations established in Chapter III do provide useful insights for design and

implementation of telemetry systems, an ideal characterization of traffic structure would provide for modeling the joint distribution of multiple spatial and temporal aspects. For example, such combined characterization might model the distribution of observed source IP addresses, observed destination IP addresses, and the number of packets sent from each source to each destination in each time window. In addition to potentially deeper insights for telemetry system design, such characterizations of traffic structure could also enable improvements in privacy-preserving realistic packet-level traffic synthesis [177]. Unfortunately, the particular joint distributions involved in network traffic structure are challenging to model under current state-of-the-art in statistics because *(i)* the spatial structure is relatively stable with a long-term memory for which particular regions are active and *(ii)* the marginal spatial and temporal distributions exhibit fundamentally different scaling behaviors (multifractal and monofractal respectively). As a result, work in this direction requires close collaboration with statistical theoreticians in order to develop the prerequisite models and analytic techniques.

6.2 Future Innovations in Designs and Algorithms for Query Processors

The designs and algorithms described in Chapter IV all suppose a fixed rhythm where traffic is monitored for a certain duration of time (referred to as an "epoch"), then all results are collected and acted upon. The the number of results is large (*e.g.,* due to a large number of traffic groups monitored), reporting all results after each epoch in this manner leads to highly bursty results traffic which is challenging to transport and process efficiently. Moreover, although fixed-duration epochs work well for the typical case where the number of groups (and hence the number of results collected) has more uniform structure (as suggested by the initial result discussed in § 3.1), the atypical cases where telemetry results are most critical (*e.g.,* to detect

and respond to attacks) are likely to have less uniform structure. As a result, one potential approach to leveraging less predictable time-domain behaviors for future work would be to abandon the strict fixed-duration epoch rhythm and investigate approaches to either dynamically varying the epoch duration (*e.g.,* in response to observed changes in number of groups) or abandoning the fixed time-window design entirely and investigating event-based, flow-based, or session-based traffic queries. More flexible query models have already been proposed in other time series analysis contexts (*e.g.,* quantitative regular expressions [108, 27] and associated automata models [28, 26]) and can potentially be leveraged for traffic monitoring [180].

6.3 Future Traffic Monitoring Applications

Although in this dissertation we only consider application of the iterative refinement methods described in § 4.2 to the task of detecting volumetric DDoS attack traffic (§ 5.2), these methods can potentially be applied to a wide range of event detection tasks. For example, security events that have prominent network-level signatures like brute-force attacks or port-scanning can be detected by a more-or-less direct application of these methods. Performance events like transient latency induced by microbursts [47] may be more challenging since the traffic signatures they produce are likely harder to detect under prefix-level aggregation and hence may require non-trivial modification of the classification model set up or refinement algorithms.

APPENDIX A

DATASETS

The content of this appendix is intended to be published with co-authors Ramakrishnan Durairajan, Reza Rejaie, Arpit Gupta, and Walter Willinger along with content from § 3.2. Walt O'Connor assisted with collecting metrics about traces.

Our efforts to assemble our own collection of measured Internet traffic traces are driven by three requirements. First, to ensure the relevance of our work for today's Internet, we require our traces to reflect "modern" Internet traffic (*e.g.,* recorded within the last ten years). Second, following [60] and using the term "traffic invariant" to mean some facet of behavior of traffic which has been empirically shown to hold in a very wide range of network settings or environments, to examine whether or not an empirical property such as the multifractal structure of observed IP addresses in measured network traffic can be viewed as such an "invariant", we require the different traces to be collected from different network locations (e.g., backbone links vs. access links) and over periods of time that span a few years. Lastly, to further examine such invariants and their presence/absence in particular portions of network traffic, we also want our collection to include traces that differ by type of traffic (*e.g.,* Darknet vs generic).

To this end, we amassed a collection of publicly available datasets, alongside a number of privately collected datasets recorded from different university campus border switches (*e.g.,* University of Oregon, UCSB) and other datasets that were provided by a third party (*e.g.,* Darknet traffic). These datasets are shown in Table A.1 where the different traces are grouped into backbone-link traffic traces, access-link traffic traces, and Darknet traffic traces. In particular, CAIDA refers to CAIDA's uni-directional traffic traces which were recorded from an Internet

backbone link in 2019 [4] and which are anonymized using CryptoPan prefix-preserving anonymization [7]; MAWI refers to (bi-directional) traffic traces from the MAWILab traffic anomaly archive [61] that were recorded at the border between the WIDE project and its parent ISP, span a period of six years (2015-2021) and are anonymized with tcpdpriv [14]; UO (UCSB) refers to captures of network traffic at the edge of the University of Oregon's (University of California Santa Barbara's) campus network (data from both campus networks were anonymized prior to analysis using the prefix-preserving anonymization method CryptoPan [7], thus ensuring that running our analysis of the observed IP addresses before and after anonymization yields identical results); and Miscellaneous refers to Darknet traces collected by Merit Network, Inc. during 2021-2023.

Table A.2 shows the results of applying the method of moments (see § 3.2.3) to estimate the three generalized dimensions for each trace in our repository. These quantities refine the binary decision "multifractal: yes or no" that results from an application of the method of moments (i.e., Step 3.1 or Step 3.2) and produces a "yes" for all the listed traces. Despite some variation in these quantities across the different traces, they consistently provide evidence for multifractal scaling (*e.g.,* all dimensions less than one), in stark contrast to their counterparts for the synthetic monofractal trace UNIFORM100 (see Table 3 in Section 4.2). Also note these estimates are numerically computed and the existing statistical theory does not provide techniques for computing meaningful confidence intervals for the computed values. As such, more precise values of the reported results would have little meaning and therefore we only show them rounded to one decimal.

ToI/ToI Instances	Date	Duration	Type	IP Count	Packet Count
ToI: Backbone-link (uni-directional)					
CAIDA-dir-A	2019-01-17 5:00AM	900s	pcap	2,381,306	566M
CAIDA-dir-B	2019-01-17 5:00AM	900s	pcap	3,863,987	1.23B
ToI: Backbone-link (bi-directional)					
MAWI-20150202	2015-02-02 2:00PM	900s	pcap	5,571,625	99M
MAWI-20150710	2015-07-10 2:00PM	900s	pcap	4,415,599	191M
MAWI-20151002	2015-10-08 2:00PM	900s	pcap	4,818,370	135M
MAWI-20180316	2018-03-16 2:00PM	900s	pcap	4,567,614	69M
MAWI-20180807	2018-08-07 2:00PM	900s	pcap	4,635,311	73M
MAWI-20181107	2018-11-07 2:00PM	900s	pcap	4,677,191	80M
MAWI-20190901	2019-09-01 2:00PM	900s	pcap	4,689,835	87M
MAWI-20210110	2021-01-10 2:00PM	900s	pcap	265,794	52M
MAWI-20210614	2021-06-14 2:00PM	900s	pcap	180,532	74M
MAWI-20211212	2021-12-12 2:00PM	900s	pcap	149,572	55M
ToI: Access-link (UCSB) (incoming)					
UCSB-20220428	2022-04-28 12:15	900s	pcap	194,498	1.02B
UCSB-20220921	2022-09-21 19:25	900s	pcap	112,164	1.05B
UCSB-20221205	2022-12-05 20:15	900s	pcap	186,575	1.1B
ToI: Access-link (UO) (incoming)					
UO-20181106	2018-11-16 00:00	1 day	netflow	1,805,085	9B
UO-20190829	2019-08-29 00:00	1 day	netflow	1,497,311	12B
UO-20200213	2020-02-13 00:00	1 day	netflow	786,607	36B
UO-20211106	2021-11-06 00:00	1 day	netflow	1,288,775	26B
UO-20220517	2022-05-17 00:00	1 day	netflow	668,817	23B
ToI: Miscellaneous					
Darknet-2023-6am	2023-03-31 06:00	1 hour	pcap	*	*
Darknet-2023-2pm	2023-03-31 14:00	1 hour	pcap	*	*
Darknet-2023-8pm	2023-03-31 20:00	1 hour	pcap	*	*

Table A.1. Our collection of recent traffic traces. (*We omit IP and packet counts for the Darknet-2023 datasets to respect privacy of the data holder.)

ToI Instances	D_0 (fractal dimension)	D_1 (information dimension)	D_2 (correlation dimension)	$\tau(1) = 0$?
CAIDA-dir-A	0.8	0.8	0.7	✓
CAIDA-dir-B	0.5	0.7	0.7	✓
MAWI-20150202	0.6	0.9	0.6	✓
MAWI-20150710	0.6	0.9	0.6	✓
MAWI-20151008	0.6	0.9	0.6	✓
MAWI-20180316	0.6	0.9	0.6	✓
MAWI-20180807	0.6	0.9	0.6	✓
MAWI-20181107	0.6	0.9	0.6	✓
MAWI-20190901	0.6	0.9	0.6	✓
MAWI-20210110	0.5	0.6	0.5	✓
MAWI-20210614	0.5	0.6	0.4	✓
MAWI-20211212	0.5	0.5	0.4	✓
UCSB-20220428	0.5	0.6	0.5	✓
UCSB-20220921	0.5	0.5	0.5	✓
UCSB-20221205	0.5	0.6	0.5	✓
UO-20181116	0.6	0.7	0.6	✓
UO-20190829	0.5	0.7	0.5	✓
UO-20200213	0.5	0.7	0.5	✓
UO-20211106	0.5	0.7	0.5	✓
UO-20220517	0.5	0.7	0.5	✓
Darknet-2023-6am	0.5	*	0.4	✓
Darknet-2023-2pm	0.5	*	0.4	✓
Darknet-2023-8pm	0.5	*	0.4	✓

Table A.2. Multifractal analysis results for all traffic traces listed in A.1. (*We only had access to Darknet-2023 for a limited duration and did not have an opportunity to compute D_1 for this dataset.)

APPENDIX B

STEP-BY-STEP PROCEDURE FOR APPLYING THE METHOD OF MOMENTS

The content of this appendix is intended to be published with co-authors Ramakrishnan Durairajan, Reza Rejaie, Arpit Gupta, and Walter Willinger along with content from § 3.2.

Consider the multi-resolution decomposition $\{P_l : l = 0, 2, \ldots, 32\}$ of the unit interval $[0, 1)$ in 1D where for $0 \leq l \leq 32$, P_l is the l-th dyadic partition of $[0, 1)$; i.e., $P_l = \{E_{l,i} = [i2^{-l}, (i + 1)2^{-l}); i = 0, 1, \ldots, 2^l - 1\}$. Let μ_A be the finite measure on $[0, 1)$ that counts for each subset $C \in [0, 1)$ the number of observed IP addresses that fall into C. Multifractal analysis of real-world data by means of the method of moment entails applying the following steps:

Step 1: For different values of q, compute for each $0 \leq l \leq 32$, the partition functions $Z(l, q) = \sum_i \mu_A(E_{l,i})^q)$ where the sum is taken over all non-empty $E_{l,i}$

Step 2: For each q-values, plot $\log Z(l, q)$ vs. l $(0 \leq l \leq 32)$, fit a straight line over some range of sufficiently large l-values, and compute the estimate $\widehat{\tau(q)}$ of the structure function $\tau(q)$ as the slope of this straight line.

Step 3.1: Interpret a linear inferred structure function $\widehat{\tau(q)}$ (as a function of q) as statistical evidence that the data is consistent with mono-fractal scaling but inconsistent with multifractal scaling.

Step 3.2: If based on Step 3.1, the data is found to be consistent with multifractal scaling, then compute estimators of the **generalized dimensions**

D_0, D_1, and D_2 (we use the convention $r = 2^{-l}$) by setting

$$\widehat{D_0} := \lim_{r\to 0}(-\frac{\log Z(r,0)}{\log r}) = \widehat{\tau(0)}, \tag{B.1}$$

$$\widehat{D_1} := \lim_{r\to 0}\frac{-\sum_i \mu(E_{l,i})\log\mu(E_{l,i})}{\log r}, \tag{B.2}$$

$$\widehat{D_2} := \lim_{r\to 0}(\frac{\log Z(r,2)}{\log r}) = -\widehat{\tau(2)}, \tag{B.3}$$

APPENDIX C

PREFIX-LEVEL MACHINE LEARNING

The content of this appendix is adapted from [117].

This appendix describes the per-prefix risk model trained to report when a monitored prefix contains one or more volumetric DDoS attack sources in ZAPDOS.

Why use machine learning (ML)? To understand why we use an ML-based per-prefix risk model, consider a simpler method that decides if a prefix is suspicious based on applying a threshold to a single metric such as the difference between DNS requests and DNS responses as proposed at source-level in [103]. A key challenge with using a threshold in prefix-level detection is that the baseline volume of traffic changes drastically with prefix length and differently in different prefixes (*i.e.,* due to clustering [31, 90, 91]). A non-linear model trained on features from a diverse range of prefixes at different lengths as used in ZAPDOS, on the other hand, can capture the complex correlation between prefix length, traffic volume and other more descriptive features (*e.g.,* inter-packet gap statistics).

The challenge of appropriate data. As with other ML-based methods, the success of ZAPDOS's model depends first and foremost on the availability of a *large* and *representative* dataset of observed feature vectors and labels from both classes. However, limited availability of appropriate data is a persistent and well-established problem when applying ML classification techniques to network security tasks [148, 96]. In developing ZAPDOS, this problem is particularly pronounced because the model must be trained over observations that capture the realistic blending of attack and benign features under prefix aggregation. Datasets and methodologies used in prior efforts are insufficient because they either contain no benign traffic [5, 141, 52], lack high-confidence labels [175, 163], or do not provide

realistic address-space distributions [143, 103]. To illustrate, consider for example the CIC datasets [143] used to both train and evaluate several proposed approaches to classifying volumetric DDoS attack flows [54, 29]. When viewed at the five-tuple flow level, these datasets are quite large, containing millions of flows. However, as shown in Table C.1, when viewed at the source address or source address prefix level, their size quickly collapses to an extremely small number of distinct observations. The largest (ISCX'12) contains 14 attack sources distributed in 6 /16 prefixes.

Dataset	**# Benign**				**# Attack**			
	/8	**/16**	**/24**	**/32**	**/8**	**/16**	**/24**	**/32**
CAIDA ('07) [5]	0	0	0	0	117	4 k	8.7 k	9 k
ISCX ('12) [145]	123	1590	2041	2129	6	6	9	14
Booters ('15) [141]	0	0	0	0	42	961	3 k	4.4 k
Mirai ('16) [21]	0	0	0	0	162	3.5 k	9.8 k	10 k
CIC ('17) [143]	156	922	2125	3432	1	1	1	1
CSECIC ('18) [11]	1	1	6	446	2	4	10	10
MAWILab ('19) [61]	211	30 k	3.3 m	5.3 m	0	0	0	0
CAIDA ('19) [4]	250	27 k	323 k	1.3 m	0	0	0	0
Proposed "data-fusion" method	216	30 k	3.2 m	4.8 m	179	7 k	45 k	50 k

Table C.1. Number of distinct attack and benign prefixes in datasets commonly used for training and testing ML-based approaches to DDoS traffic detection.

C.1 Data-fusion for Realistic Prefix-level Features

Given the insufficiency of existing datasets, we develop a novel data-fusion approach to training and evaluating ZAPDOS. As shown in Figure C.1, our method involves several real-life and synthetic data sources which provide realistic attack address distributions as well as packet-level data. This method allows us to generate multiple, independent attack scenarios, each with realistic, disjoint sets of attack and benign sources and hence representative blending of features under prefix aggregation. In particular, we take extra caution to ensure proper separation of prefix-level features between training and testing sets to avoid model overfitting. Overall we generate

84 distinct attacks with varying numbers of sources and attack vectors, 42 training scenarios and 42 test scenarios, using the following steps.

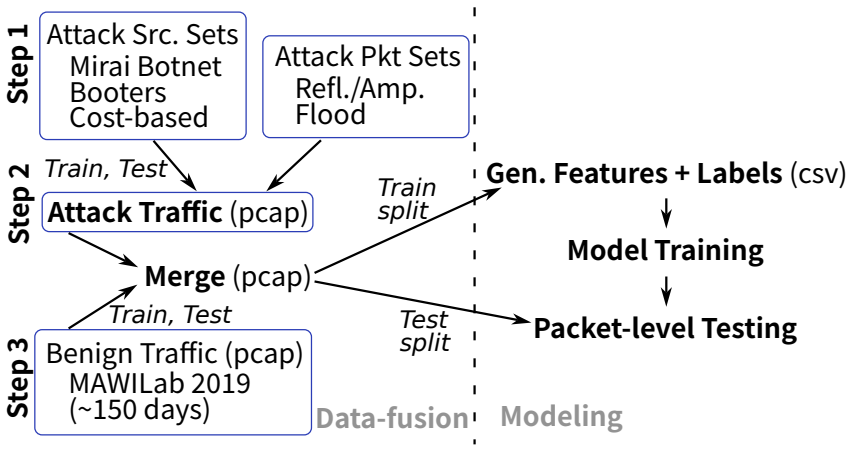


Figure C.1. Diagram of dataset preparation and modeling methodology used in ZAPDOS.

Step 1: Generate attack source sets. We use three methods of obtaining realistic sets of attack source addresses: Mirai botnet [21] (∼18 M distinct sources), Booters dataset [141] (∼18 K distinct sources), and a synthetic cost-based method. For our training data, we extract disjoint sets with varying numbers of sources from the Mirai data set (for simplicity we use sets of 500, 1k, 2k, 5k, 10k, 20k, and 50k sources). Booters sources are used in our evaluation to demonstrate how ZAPDOS can generalize to entirely unseen source distributions.

We also develop a *cost-based method* to understand how ZAPDOS reacts when the adversary invests effort to infer benign prefixes during the pre-attack phase and spoofs attack sources to come from these prefixes. In particular, we define cost for the attacker in terms of how many prefix bits ℓ attack sources share with benign sources. Then, given a set of benign sources, to generate attack sources for a particular cost ℓ, we identify the set of $/\ell$ prefixes that share a $/(\ell-1)$ prefix with a benign source, then select attack sources uniformly at random in these identified prefixes. In this

way the clustering of attack sources is closer to (farther from) benign sources for larger (smaller) values of ℓ.

Step 2: Generate attack traffic. Next, we generate packet-level attack traffic for each enumerated attack source based on attack vectors. Table C.2 shows the six different attack vectors as well as packet-level sources or parameters considered in this work. We fix the target bit rate of the aggregate attack traffic (*e.g.,* 1 Gbps) and the number of attack sources to compute the per-source bit rate which determines the arrival time of attack packets. Here, an attack *A1* with the same target bit rate as *A2*, but with more attack sources, will have lower per-source bit rates (and vice versa). To mimic a real attack [141], each source generates packets at a constant bit rate with a uniform $\pm 1\%$ variance around the target per-source rate.

Type	**Attack Vector**	**Packet Source/Type**
Refl./Amp.	DNS	"Booter1" [141]
	NTP	CIC DDoS "day two" [144]
	SSDP	Sucuri analysis [119]
Flood	SYN	TCP with SYN flag set
	ICMP	ICMP Echo request
	UDP	UDP random payload

Table C.2. Attack vectors generated for evaluation of ZAPDOS and the sources or types of attack packets used.

Step 3: Merge attack and benign traffic. Finally, we interleave the packets of each generated attack with benign traffic from the 2019 MAWILab dataset [61] (preserving the relative packet arrival times of both traces). The MAWILab dataset provides packet-level data for a large number of benign traffic excerpts allowing us to assign a unique excerpt to each generated attack scenario. Although using a single supplier of benign traffic does not allow us to reason about a trained model's ability to generalize to other network settings, our data-fusion methodology can be

replicated with any other source of benign traffic in order to produce datasets and models tailored for particular networks.

C.2 Modeling Setup

Feature selection. To train the classification model used in ZAPDOS, we compute the features described in Table C.3 for each prefix in each of the training attack scenarios. At a high level, we carefully selected features that on the one hand intuitively capture differences in traffic patterns between attack and benign behaviors and on the other can be computed in switch hardware at line rate. Though similar features have been used for flow-level attack classification [161, 112, 152, 54], we reiterate that in ZAPDOS, all features are aggregate over all traffic from the prefix being classified. As a result, we only require maintaining feature state during each epoch for a small, limited number of prefixes using queries described in § 4.2.1 instead of over all flows as in prior efforts.

	Feature	Description
	Prefix length	Number of leading bits defining prefix.
Directional	Bytes from	Number of bytes from this prefix.
	Bytes to	Number of bytes to this prefix.
	Packets from	Number of packets from this prefix.
	Packets to	Number of packets to this prefix.
Statistical	min/max/ave. len.	Minimum, maximum, and moving average of length of packets from or to this prefix.
	min/max/ave. IPG	Minimum, maximum, and moving average of time between consecutive packets from or to this prefix.
	Last active	Number of epochs since any packets were last observed from or to this prefix.
Generic	$rrDiff$	Maximum (responses - requests) over a number of known amplification protocols (e.g., DNS, NTP, etc.), see § C.3.

Table C.3. List of features computed each epoch for each monitored prefix for use in ZAPDOS's model.

Model selection. We use a random forest (RF) model [43, 51] with 500 trees and other defaults as set in [15] to predict if a prefix sources attack traffic based on

the observed features. We chose to use RF due to its observed high accuracy and fast training and classification times compared to more complex methods like neural networks.

Weighted training. Initial experiments suggested that the per-prefix false-negative and false-positive rates of our model do not directly correspond to the error rates of ZAPDOS's end-to-end iterative refinement approach. In particular, we found that false-negative decisions made at shorter prefix lengths had a disproportional impact on the overall false-negative rate because they caused the refinement process to miss large prefixes potentially containing many attack sources. To counteract this effect, when training our model, we use a weighted sample of our training set which contains more observations of attack prefixes at shorter prefix lengths (*e.g.,* less than /16) compared to longer prefix lengths. This weighting of the training set causes the model to be more suspicious (*i.e.,* make more false-positive decisions) at shorter prefix lengths, making it more likely that ZAPDOS will continue zooming in and not miss large regions that include attack sources.

C.3 Request-Response Difference

In this section we describe the generic $rrDiff$ feature which summarizes multiple well-known signatures at the prefix-level. Intuitively, $rrDiff$ captures patterns where attack traffic tends to have a larger number of packets of one class (*e.g.,* responses) compared to another class (*e.g.,* requests). (Note that we will refer to these two classes as "responses" and "requests" in the following.)

In particular, for each prefix p and each attack vector v, we compute the total number of request packets ($req_{p,v}$) and the total number of response packets ($resp_{p,v}$), respectively. Table C.4 shows the set of response and request entries for the attack vectors considered in this work along with details about how we classify

packets as being requests or responses for each vector. Given these sums that the switch hardware computes for a given prefix p, we consider the feature $rrDiff_p$ that summarizes the degree of response-request imbalance in that prefix and is defined as

$$rrDiff_p = \max_v (resp_{p,v} - req_{p,v})$$

Intuitively, the value of $rrDiff_p$ will be high if p sources attack traffic and close to zero if p sources only benign traffic.

Protocol	**Request**	**Response**
DNS refl.	UDP to port 53	UDP from port 53
NTP refl.	UDP to port 123	UDP from port 123
SSDP refl.	UDP to port 1900	UDP from port 1900
SYN flood[1]	TCP, SYN+ACK set	TCP, only SYN set

Table C.4. List of currently considered reflection vectors.

Feature	**IncNodePurity**
prefixLength	759.0381
bytesFrom	41200.3716
bytesTo	2066.1812
respReqDiff	6761.4385
lastActiveDiff	1010.9086
minIPG	2039.5534
maxIPG	4436.0248
aveIPG	1274.9780
pktsFrom	17092.2162
pktsTo	1558.1276
minLen	0.0000
maxLen	5569.0806
aveLen	11052.1292

Table C.5. Feature importance of the trained random forest model used in ZAPDOS.

[1]Note that SYN floods are not technically a reflection attack, but we can still capture their asymmetry with our response/request difference feature.

APPENDIX D

GLOSSARY OF SELECTED ABBREVIATIONS

The following is a list of commonly used abbreviations in this dissertation.

- **ALU.** Arithmetic Logic Unit.
- **API.** Application Programming Interface.
- **ASIC.** Application-Specific Integrated Circuit.
- **BDD.** Binary Decision Diagram.
- **CDN.** Content Delivery Network.
- **CIDR.** Classless Inter-Domain Routing.
- **CPU.** Central-Processing Unit.
- **CSP.** Cloud Service Provider.
- **CV.** Coefficient of Variation.
- **DDoS.** Distributed Denial of Service (a type of network-based attack).
- **DMA.** Direct Memory Access.
- **DNS.** Domain Name System.
- **DSL.** Domain-Specific Language.
- **ECMP.** Equal-Cost Multi-Path (a routing policy that distributes traffic over multiple links).
- **EWMA.** Exponentially Weighted Moving Average.
- **FPR.** False-Positive Rate.
- **FNR.** False-Negative Rate.
- **IANA.** Internet Assigned Numbers Authority.
- **ICANN.** Internet Corporation for Assigned Names and Numbers.
- **ICMP.** Internet Control Message Protocol.

- **IDS.** Intrusion Detection System.
- **IP.** Internet Protocol.
- **ISP.** Internet Service Provider.
- **JSON.** JavaScript Object Notation.
- **REST.** REpresentational State Transfer.
- **RIR.** Regional Internet Registry.
- **RPC.** Remote Procedure Call.
- **SDN.** Software-Defined Networking.
- **SRAM.** Static Random Access Memory.
- **SVK.** System Verification Kit.
- **TCAM.** Ternary Content-Addressable Memory.
- **TCP.** Transmission Control Protocol.
- **UDP.** User Datagram Protocol.

REFERENCES CITED

[1] Apache flink: Stateful computations over data streams. https://flink.apache.org/. Accessed: 2024-06.

[2] BCM56275 Gb/s Programmable Multilayer Switch Product Brief. https://docs.broadcom.com/doc/56275-PB. Accessed: 2024-06.

[3] Bulk whois data - american registry for internet numbers. https://www.arin.net/reference/research/bulkwhois/. Accessed: 2024-01.

[4] The CAIDA UCSD anonymized Internet traces dataset - 2019. https://www.caida.org/data/monitors/passive-equinix-nyc.xml. Accessed: 2024-06.

[5] The CAIDA UCSD "DDoS attack 2007" dataset. http://www.caida.org/data/passive/ddos-20070804_dataset.xml. Accessed: 2022.

[6] COIN-OR Branch-and-cut MIP solver. https://zenodo.org/badge/latestdoi/30382416. Accessed: 2024-06.

[7] Cryptopan. https://en.wikipedia.org/wiki/Crypto-PAn. Accessed: 2024-06.

[8] Explore the power of intel® intelligent fabric processors. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html. Accessed: 2024-06.

[9] Haskell language. https://www.haskell.org/. Accessed 2023-12-05.

[10] IANA IPv4 address space registry. https://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml. Accessed: 2024-01.

[11] IDS 2018. https://www.unb.ca/cic/datasets/ids-2018.html. Accessed: 2023-06-22.

[12] IPv6 global unicast address assignments. https://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xhtml. Accessed: 2024-01.

[13] ONRG: DynATOS. https://onrg.gitlab.io/projects/dynatos/. Accessed: 2024-06.

[14] Program for elimination confidential information from traces. https://fly.isti.cnr.it/software/tcpdpriv/. Accessed: 2024-02.

[15] randomForest: Breiman and cutler's random forests for classification and regression. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf. Accessed: 2022.

[16] sonata-queries/sonata-queries. https://github.com/sonata-queries/sonata-queries. Accessed: Nov. 2022.

[17] Tcpreplay - Pcap editing and replaying utilities. https://tcpreplay.appneta.com/. Accessed: 2024-06.

[18] Trident3-X4 / BCM56470 Series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56470-series. Accessed: 2024-06.

[19] unagi-chan: Fast concurrent queues with a chan-like api, and more. https://hackage.haskell.org/package/unagi-chan. Accessed 2023-12-05.

[20] The zeek network security monitor. https://zeek.org/. Accessed: 2024-04-21.

[21] FRGP (www.frgp.net) continuous flow dataset, IMPACT ID: USC-LANDER/Mirai-FRGP-scanning-20160908/rev10326. provided by the USC/LANDER project (http://www.isi.edu/ant/lander), traces taken 2016-09-08 to 2016-10-31.

[22] Abry, P., and Veitch, D. Wavelet analysis of long-range-dependent traffic. *IEEE transactions on information theory 44*, 1 (1998), 2–15.

[23] Agarwal, A., Liu, Z., and Seshan, S. HeteroSketch: Coordinating network-wide monitoring in heterogeneous and dynamic networks. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 719–741.

[24] Alcoz, A. G., Strohmeier, M., Lenders, V., and Vanbever, L. Aggregate-based congestion control for pulse-wave ddos defense. In *Proceedings of the ACM SIGCOMM Conference* (2022), pp. 693–706.

[25] Alshnakat, A., Lundberg, D., Guanciale, R., Dam, M., and Palmskog, K. Hol4p4: semantics for a verified data plane. In *Proceedings of the 5th International Workshop on P4 in Europe* (2022), pp. 39–45.

[26] Alur, R., Fisman, D., Mamouras, K., Raghothaman, M., and Stanford, C. Streamable regular transductions. *Theoretical Computer Science 807* (2020), 15–41.

[27] Alur, R., Fisman, D., and Raghothaman, M. Regular programming for quantitative properties of data streams. In *European Symposium on Programming* (2016), Springer, pp. 15–40.

[28] Alur, R., Mamouras, K., and Stanford, C. Modular quantitative monitoring. *Proceedings of the ACM on Programming Languages 3*, POPL (2019), 1–31.

[29] Asad, M., Asim, M., Javed, T., Beg, M. O., Mujtaba, H., and Abbas, S. Deepdetect: detection of distributed denial of service attacks using deep learning. *The Computer Journal 63*, 7 (2020), 983–994.

[30] Avin, C., Ghobadi, M., Griner, C., and Schmid, S. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 4*, 1 (2020), 1–29.

[31] Barford, P., Nowak, R., Willett, R., and Yegneswaran, V. Toward a model for source addresses of internet background radiation. In *Proceedings of the Passive and Active Measurement Conference* (2006), Citeseer.

[32] Barradas, D., Santos, N., Rodrigues, L., Signorello, S., Ramos, F. M., and Madeira, A. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *Proceedings of the Network and Distributed System Security Symposium* (2021).

[33] Basat, R. B., Chen, X., Einziger, G., and Rottenstreich, O. Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking 28*, 3 (2020), 1172–1185.

[34] Beran, J., Sherman, R., Taqqu, M. S., and Willinger, W. Long-range dependence in variable-bit-rate video traffic. *IEEE Transactions on communications 43*, 2/3/4 (1995), 1566–1579.

[35] Bernier, D. Rfc 9378: In situ operations, administration, and maintenance (ioam) deployment, 2023.

[36] Bhatia, R., Gupta, A., Harrison, R., Lokshtanov, D., and Willinger, W. Dynamiq: Planning for dynamics in network streaming analytics systems. *arXiv preprint arXiv:2106.05420* (2021).

[37] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (1970), 422–426.

[38] Borders, K., Springer, J., and Burnside, M. Chimera: A declarative language for streaming network traffic analysis. In *Proceedings of the USENIX Security Symposium* (2012), pp. 365–379.

[39] Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 87–95.

[40] Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review 43*, 4 (2013), 99–110.

[41] Braverman, V., and Ostrovsky, R. Zero-one frequency laws. In *Proceedings of the forty-second ACM symposium on Theory of computing* (2010), pp. 281–290.

[42] Braverman, V., and Ostrovsky, R. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization* (2013), Springer, pp. 58–70.

[43] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[44] Chao, A., and Chiu, C.-H. Species richness: estimation and comparison. *Wiley StatsRef: Statistics Reference Online* (2014), 1–26.

[45] Chao, A., and Lin, C.-W. Nonparametric lower bounds for species richness and shared species richness under sampling without replacement. *Biometrics 68*, 3 (2012), 912–921.

[46] Charikar, M., Chen, K., and Farach-Colton, M. Finding frequent items in data streams. In *Proceedings of the International Colloquium on Automata, Languages and Programming* (2002), Springer, pp. 693–703.

[47] Chen, X., Feibish, S. L., Koral, Y., Rexford, J., and Rottenstreich, O. Catching the microburst culprits with snappy. In *Proceedings of the ACM Workshop on Self-Driving Networks* (2018), pp. 22–28.

[48] Chen, X., Landau-Feibish, S., Braverman, M., and Rexford, J. Beaucoup: Answering many network traffic queries, one memory update at a time. In *Proceedings of the ACM SIGCOMM Conference* (2020), pp. 226–239.

[49] Ciaccia, F., Romero, I., Arcas-Abella, O., Montero, D., Serral-Gracià, R., and Nemirovsky, M. Sabes: Statistical available bandwidth estimation from passive tcp measurements. In *Proceedings of the IFIP Networking Conference* (2020), IEEE, pp. 743–748.

[50] Cormode, G., and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms 55*, 1 (2005), 58–75.

[51] Cutler, A., Cutler, D. R., and Stevens, J. R. Random forests. In *Ensemble machine learning*. Springer, 2012, pp. 157–175.

[52] da Silveira Ilha, A., Lapolli, Â. C., Marques, J. A., and Gaspary, L. P. Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation. *IEEE Transactions on Network and Service Management* (2020).

[53] Doenges, R., Arashloo, M. T., Bautista, S., Chang, A., Ni, N., Parkinson, S., Peterson, R., Solko-Breslin, A., Xu, A., and Foster, N. Petr4: formal foundations for p4 data planes. *Proceedings of the ACM on Programming Languages 5*, POPL (2021).

[54] Doriguzzi-Corin, R., Millar, S., Scott-Hayward, S., Martinez-del Rincon, J., and Siracusa, D. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management* (2020).

[55] Ekelin, S., Nilsson, M., Hartikainen, E., Johnsson, A., Mangs, J.-E., Melander, B., and Bjorkman, M. Real-time measurement of end-to-end available bandwidth using kalman filtering. In *Proceedings of the IEEE/IFIP Symposium on Network Operations and Management* (2006), IEEE, pp. 73–84.

[56] Eppstein, D., Goodrich, M. T., Uyeda, F., and Varghese, G. What's the difference? efficient set reconciliation without prior context. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 218–229.

[57] Erramilli, A., Narayan, O., Neidhardt, A., and Saniee, I. Performance impacts of multi-scaling in wide area TCP/IP traffic. In *Proceedings of IEEE INFOCOM* (2000), vol. 1, IEEE, pp. 352–359.

[58] Feldmann, A., Gilbert, A. C., and Willinger, W. Data networks as cascades: Investigating the multifractal nature of internet wan traffic. *ACM SIGCOMM Computer Communication Review 28*, 4 (1998), 42–55.

[59] Feng, Y., Chen, Z., Song, H., Xu, W., Li, J., Zhang, Z., Yun, T., Wan, Y., and Liu, B. Enabling in-situ programmability in network data plane: From architecture to language. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 635–649.

[60] Floyd, S., and Paxson, V. Difficulties in simulating the internet. *IEEE/ACM Transactions on networking 9*, 4 (2001), 392–403.

[61] Fontugne, R., Borgnat, P., Abry, P., and Fukuda, K. MAWILab: Combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies* (2010).

[62] Fornasa, M., and Maresca, M. Passive access capacity estimation through the analysis of packet bursts. In *Proceedings of the ICST Conference on Access Networks* (2011), Springer, pp. 83–99.

[63] Gangam, S., Sharma, P., and Fahmy, S. Pegasus: Precision hunting for icebergs and anomalies in network flows. In *Proceedings of IEEE INFOCOM* (2013), pp. 1420–1428.

[64] Gao, K., Sun, C., Wang, S., Li, D., Zhou, Y., Liu, H. H., Zhu, L., and Zhang, M. Buffer-based end-to-end request event monitoring in the cloud. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 829–843.

[65] Geravand, S., and Ahmadi, M. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks 57*, 18 (2013), 4047–4064.

[66] Gibbons, P. B. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB* (2001), vol. 1, pp. 541–550.

[67] Gilbert, A. C., Willinger, W., and Feldmann, A. Scaling analysis of conservative cascades, with applications to network traffic. *IEEE Transactions on Information Theory 45*, 3 (1999), 971–991.

[68] Gombos, G., Mouw, M., Laki, S., Papagianni, C., and De Schepper, K. Active queue management on the tofino programmable switch: The (dual) pi2 case. In *Proceedings of the IEEE International Conference on Communications* (2022), IEEE, pp. 1685–1691.

[69] Goodrich, M. T., and Mitzenmacher, M. Invertible bloom lookup tables. In *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing* (2011), IEEE, pp. 792–799.

[70] Gupta, A., Harrison, R., Canini, M., Feamster, N., Rexford, J., and Willinger, W. Sonata: Query-driven streaming network telemetry. In *Proceedings of the ACM SIGCOMM Conference* (2018), pp. 357–371.

[71] Harrison, R., Cai, Q., Gupta, A., and Rexford, J. Network-wide heavy hitter detection with commodity switches. In *Proceedings of the Symposium on SDN Research* (2018), pp. 1–7.

[72] Heule, S., Nunkesser, M., and Hall, A. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proceedings of the 16th International Conference on Extending Database Technology* (2013), pp. 683–692.

[73] Hohn, N., Veitch, D., and Abry, P. Does fractal scaling at the ip level depend on tcp flow arrival processes? In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), pp. 63–68.

[74] Hohn, N., Veitch, D., and Abry, P. Investigating the scaling behaviour of internet flow arrivals. In *Proceedings of the International Conference on Self-Similarity and Applications* (2002), Citeseer.

[75] Hohn, N., Veitch, D., and Abry, P. Cluster processes: a natural language for network traffic. *IEEE Transactions on Signal processing 51*, 8 (2003), 2229–2244.

[76] Hohn, N., Veitch, D., and Abry, P. The impact of the flow arrival process in internet traffic. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (2003), vol. 6, IEEE, pp. VI–37.

[77] Hsu, A., Li, F., and Pearce, P. Fiat lux: Illuminating ipv6 apportionment with different datasets. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 7*, 1 (2023), 1–24.

[78] Huang, Q., Lee, P. P., and Bao, Y. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the ACM SIGCOMM Conference* (2018), pp. 576–590.

[79] Huang, Q., Sheng, S., Chen, X., Bao, Y., Zhang, R., Xu, Y., and Zhang, G. Toward nearly-zero-error sketching via compressive sensing. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2021), pp. 1027–1044.

[80] Huang, Q., Sun, H., Lee, P. P., Bai, W., Zhu, F., and Bao, Y. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the ACM SIGCOMM Conference* (2020), pp. 404–421.

[81] ICANN. Available pool of unallocated IPv4 internet addresses now completely emptied. `https://itp.cdn.icann.org/en/files/announcements/release-03feb11-en.pdf`. Accessed: 2024-01.

[82] Ivkin, N., Liberty, E., Lang, K., Karnin, Z., and Braverman, V. Streaming quantiles algorithms with small space and update time. *Sensors 22*, 24 (2022), 9612.

[83] Ivkin, N., Yu, Z., Braverman, V., and Jin, X. Qpipe: Quantiles sketch fully in the data plane. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies* (2019), pp. 285–291.

[84] Jaffard, S., Lashermes, B., and Abry, P. Wavelet leaders in multifractal analysis. In *Wavelet analysis and applications* (2007), Springer, pp. 201–246.

[85] Joshi, R., Qu, T., Chan, M. C., Leong, B., and Loo, B. T. Burstradar: Practical real-time microburst monitoring for datacenter networks. In *Proceedings of the 9th Asia-Pacific Workshop on Systems* (2018), pp. 1–8.

[86] Kang, M. S., Lee, S. B., and Gligor, V. D. The crossfire attack. In *Proceedings of the IEEE Symposium on Security and Privacy* (2013), pp. 127–141.

[87] Karami, M., Park, Y., and McCoy, D. Stress testing the booters: Understanding and undermining the business of ddos services. In *Proceedings of the 25th International Conference on World Wide Web* (2016), pp. 1033–1043.

[88] Kheradmand, A., and Rosu, G. P4k: A formal semantics of p4 and applications. *arXiv preprint arXiv:1804.01468* (2018).

[89] Kim, C., Bhide, P., Doe, E., Holbrook, H., Ghanwani, A., Daly, D., Hira, M., and Davie, B. In-band network telemetry (INT). `https://p4.org/assets/INT-current-spec.pdf`, June 2016.

[90] Kohler, E., Li, J., Paxson, V., and Shenker, S. Observed structure of addresses in ip traffic. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), pp. 253–266.

[91] Kohler, E., Li, J., Paxson, V., and Shenker, S. Observed structure of addresses in ip traffic. *IEEE/ACM Transactions on Networking 14*, 6 (2006), 1207–1218.

[92] Kumar, R., Asif, S., Lee, E., and Bustamante, F. E. Each at its own pace: Third-party dependency and centralization around the world. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 7*, 1 (2023), 1–29.

[93] Kundel, R., Blendin, J., Viernickel, T., Koldehofe, B., and Steinmetz, R. P4-codel: Active queue management in programmable data planes. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks* (2018), IEEE, pp. 1–4.

[94] Lai, Y.-K., Shih, K.-Y., Huang, P.-Y., Lee, H.-P., Lin, Y.-J., Liu, T.-L., and Chen, J. H. Sketch-based entropy estimation for network traffic analysis using programmable data plane asics. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communications Systems* (2019), IEEE, pp. 1–2.

[95] Laskin, N. Fractional poisson process. *Communications in Nonlinear Science and Numerical Simulation 8*, 3-4 (2003), 201–213.

[96] Lavinia, Y., Durairajan, R., Rejaie, R., and Willinger, W. Challenges in using ml for networking research: How to label if you must. In *Proceedings of the Workshop on Network Meets AI & ML* (2020), pp. 21–27.

[97] Leland, W. E., Taqqu, M. S., Willinger, W., and Wilson, D. V. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking 2*, 1 (1994), 1–15.

[98] Lévy Véhel, J., and Riedi, R. Fractional brownian motion and data traffic modeling: The other end of the spectrum. In *Fractals in Engineering: from theory to industrial applications* (1997), Springer, pp. 185–202.

[99] Li, Y., Gao, K., Jin, X., and Xu, W. Concerto: cooperative network-wide telemetry with controllable error rate. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems* (2020), pp. 114–121.

[100] Li, Y., Miao, R., Kim, C., and Yu, M. FlowRadar: a better NetFlow for data centers. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2016), pp. 311–324.

[101] Li, Y., Miao, R., Kim, C., and Yu, M. Lossradar: Fast detection of lost packets in data center networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies* (2016), pp. 481–495.

[102] Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., and Braverman, V. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the ACM SIGCOMM Conference* (2016), ACM, pp. 101–114.

[103] Liu, Z., Namkung, H., Nikolaidis, G., Lee, J., Kim, C., Jin, X., Braverman, V., Yu, M., and Sekar, V. Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches. In *Proceedings of the USENIX Security Symposium* (2021).

[104] Liu, Z., Zhou, S., Rottenstreich, O., Braverman, V., and Rexford, J. Memory-efficient performance monitoring on programmable switches with lean algorithms. In *Symposium on Algorithmic Principles of Computer Systems* (2020), SIAM, pp. 31–44.

[105] Lohr, S. L. *Sampling: Design and Analysis: Design And Analysis*. CRC Press, 2019.

[106] Luo, X., and Chang, R. K. On a new class of pulsing denial-of-service attacks and the defense. In *Proceedings of the Network and Distributed System Security Symposium* (2005).

[107] Mahajan, R., Bellovin, S. M., Floyd, S., Ioannidis, J., Paxson, V., and Shenker, S. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review 32*, 3 (2002), 62–73.

[108] Mamouras, K., Raghothaman, M., Alur, R., Ives, Z. G., and Khanna, S. StreamQRE: Modular specification and efficient evaluation of quantitative queries over streaming data. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (2017), pp. 693–708.

[109] Mandelbrot., B. B. Limit lognormal multifractal measures. *Frontiers of Physics: Proceedings of the Landau Memorial Conference* (1990.), 309—-340.

[110] Mannersalo, P., and Norros, I. Multifractal analysis of real atm traffic: a first look, 1997.

[111] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review 38*, 2 (2008), 69–74.

[112] Meitei, I. L., Singh, K. J., and De, T. Detection of ddos dns amplification attack using classification algorithm. In *Proceedings of the International Conference on Informatics and Analytics* (2016), pp. 1–6.

[113] Metwally, A., Agrawal, D., and El Abbadi, A. Efficient computation of frequent and top-k elements in data streams. In *Proceedings of the International Conference on Database Theory* (2005), Springer, pp. 398–412.

[114] Minton, G. T., and Price, E. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms* (2014), SIAM, pp. 669–686.

[115] Mirkovic, J., Dietrich, S., Dittrich, D., and Reiher, P. *Internet denial of service: attack and defense mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, 2004.

[116] Misa, C., Durairajan, R., Rejaie, R., and Willinger, W. Dynatos +: A network telemetry system for dynamic traffic and query workloads. *IEEE/ACM Transactions on Networking* (2024).

[117] Misa, C., Durairajan, R., Rejaie, R., and Willinger, W. Leveraging prefix structure to detect volumetric ddos attack signatures with programmable switches. In *Proceedings of the IEEE Symposium on Security and Privacy* (2024), IEEE.

[118] Misa, C., O'Connor, W., Durairajan, R., Rejaie, R., and Willinger, W. Dynamic scheduling of approximate telemetry queries. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 701–717.

[119] Montoro, R. Quick analysis of a DDoS attack using ssdp. `https://blog.sucuri.net/2014/09/quick-analysis-of-a-ddos-attack-using-ssdp.html`, 2022.

[120] Moshref, M., Yu, M., Govindan, R., and Vahdat, A. DREAM: Dynamic resource allocation for software-defined measurement. *ACM SIGCOMM Computer Communication Review 44*, 4 (2014), 419–430.

[121] Moshref, M., Yu, M., Govindan, R., and Vahdat, A. SCREAM: Sketch resource allocation for software-defined measurement. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies* (2015), p. 14.

[122] Namkung, H., Liu, Z., Kim, D., Sekar, V., and Steenkiste, P. SketchLib: Enabling efficient sketch-based monitoring on programmable switches. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 743–759.

[123] Narayana, S., Sivaraman, A., Nathan, V., Goyal, P., Arun, V., Alizadeh, M., Jeyakumar, V., and Kim, C. Language-directed hardware design for network performance monitoring. In *Proceedings of the ACM SIGCOMM Conference* (2017), pp. 85–98.

[124] Nowak, R. D. Fractal modeling and analysis of poisson processes. In *Conference Record of Thirty-Second Asilomar Conference on Signals, Systems and Computers (Cat. No. 98CH36284)* (1998), vol. 1, IEEE, pp. 727–731.

[125] Nuzman, C., Saniee, I., Sweldens, W., and Weiss, A. A compound model for tcp connection arrivals for lan and wan applications. *Computer Networks 40*, 3 (2002), 319–337.

[126] Olsen, L. A multifractal formalism. *Advances in mathematics 116*, 1 (1995), 82–196.

[127] Pagh, R., and Rodler, F. F. Cuckoo hashing. *Journal of Algorithms 51*, 2 (2004), 122–144.

[128] Pan, T., Yu, N., Jia, C., Pi, J., Xu, L., Qiao, Y., Li, Z., Liu, K., Lu, J., Lu, J., et al. Sailfish: Accelerating cloud-scale multi-tenant multi-service gateways with programmable switches. In *Proceedings of the ACM SIGCOMM Conference* (2021), pp. 194–206.

[129] Paxson, V. Bro: a system for detecting network intruders in real-time. *Computer networks 31*, 23-24 (1999), 2435–2463.

[130] Politi, M., Kaizoji, T., and Scalas, E. Full characterization of the fractional poisson process. *Europhysics Letters 96*, 2 (2011), 20004.

[131] Prechelt, L. Early stopping-but when? In *Neural Networks: Tricks of the trade*. Springer, 2002, pp. 55–69.

[132] Ran, Y., Wu, X., Li, P., Xu, C., Luo, Y., and Wang, L.-M. Equery: Enable event-driven declarative queries in programmable network measurement. In *Proceedings of the IEEE/IFIP Symposium on Network Operations and Management* (2018), IEEE, pp. 1–7.

[133] Raskutti, G., Wainwright, M. J., and Yu, B. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *The Journal of Machine Learning Research 15*, 1 (2014), 335–366.

[134] Resnick, S., Samorodnitsky, G., Gilbert, A., and Willinger, W. Wavelet analysis of conservative cascades. *Bernoulli 9*, 1 (2003), 97–135.

[135] Riedi, R. An improved multifractal formalism and self-similar measures. *Journal of Mathematical Analysis and Applications 189*, 2 (1995), 462–490.

[136] Riedi, R. H., Crouse, M. S., Ribeiro, V. J., and Baraniuk, R. G. A multifractal wavelet model with application to network traffic. *IEEE transactions on Information Theory 45*, 3 (1999), 992–1018.

[137] Rossow, C. Amplification hell: Revisiting network protocols for ddos abuse. In *Proceedings of the Network and Distributed System Security Symposium* (2014), pp. 1–15.

[138] Rudman, L., and Irwin, B. Characterization and analysis of NTP amplification based DDoS attacks. In *Proceedings of Information Security for South Africa* (2015), IEEE, pp. 1–5.

[139] Ruffy, F., Wang, T., and Sivaraman, A. Gauntlet: Finding bugs in compilers for programmable packet processing. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation* (2020), pp. 683–699.

[140] Salat, H., Murcio, R., and Arcaute, E. Multifractal methodology. *Physica A: Statistical Mechanics and its Applications 473* (2017), 467–487.

[141] Santanna, J. J., van Rijswijk-Deij, R., Hofstede, R., Sperotto, A., Wierbosch, M., Granville, L. Z., and Pras, A. Booters—an analysis of ddos-as-a-service attacks. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management* (2015), IEEE, pp. 243–251.

[142] Sengupta, S., Kim, H., and Rexford, J. Continuous in-network round-trip time monitoring. In *Proceedings of the ACM SIGCOMM Conference* (2022), pp. 473–485.

[143] Sharafaldin, I., Lashkari, A. H., and Ghorbani, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the International Conference on Information Systems Security and Privacy* (2018), vol. 1, pp. 108–116.

[144] Sharafaldin, I., Lashkari, A. H., Hakak, S., and Ghorbani, A. A. Developing realistic distributed denial of service (ddos) attack dataset and taxonomy. In *Proceedings of the International Carnahan Conference on Security Technology* (2019), IEEE, pp. 1–8.

[145] Shiravi, A., Shiravi, H., Tavallaee, M., and Ghorbani, A. A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security 31*, 3 (2012), 357–374.

[146] Shou, C., Bhatia, R., Gupta, A., Harrison, R., Lokshtanov, D., and Willinger, W. Query planning for robust and scalable hybrid network telemetry systems. *Proceedings of the ACM on Networking 2* (2024), 1–27.

[147] Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., and Rexford, J. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research* (2017), pp. 164–176.

[148] Sommer, R., and Paxson, V. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy* (2010), pp. 305–316.

[149] Sonchack, J., Aviv, A. J., Keller, E., and Smith, J. M. Turboflow: Information rich flow record generation on commodity switches. In *Proceedings of the Thirteenth EuroSys Conference* (2018), pp. 1–16.

[150] Sonchack, J., Michel, O., Aviv, A. J., Keller, E., and Smith, J. M. Scaling hardware accelerated network monitoring to concurrent and dynamic queries with *flow. In *Proceedings of the USENIX Annual Technical Conference* (2018), pp. 823–835.

[151] Song, C. H., Kannan, P. G., Low, B. K. H., and Chan, M. C. Fcm-sketch: generic network measurements with data plane support. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies* (2020), pp. 78–92.

[152] Stiawan, D., Idris, M. Y. B., Bamhdi, A. M., Budiarto, R., et al. Cicids-2017 dataset feature analysis with information gain for anomaly detection. *IEEE Access 8* (2020), 132911–132921.

[153] Studer, A., and Perrig, A. The coremelt attack. In *Proceedings of the European Symposium on Research in Computer Security* (2009), Springer, pp. 37–52.

[154] Tammana, P., Agarwal, R., and Lee, M. Cherrypick: Tracing packet trajectory in software-defined datacenter networks. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research* (2015), pp. 1–7.

[155] Tammana, P., Agarwal, R., and Lee, M. Simplifying datacenter network debugging with PathDump. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation* (2016), pp. 233–248.

[156] Tammana, P., Agarwal, R., and Lee, M. Distributed network monitoring and debugging with SwitchPointer. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2018), pp. 453–456.

[157] Tang, L., Huang, Q., and Lee, P. P. Spreadsketch: Toward invertible and network-wide detection of superspreaders. In *Proceedings of IEEE INFOCOM* (2020), IEEE, pp. 1608–1617.

[158] Teixeira, R., Harrison, R., Gupta, A., and Rexford, J. Packetscope: Monitoring the packet lifecycle inside a switch. In *Proceedings of the Symposium on SDN Research* (2020), pp. 76–82.

[159] Ting, D. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data* (2018), pp. 1129–1140.

[160] Toh, A., Vij, A., and Pasha, S. Azue DDoS protection—2021 Q3 and Q4 DDoS attack trends. `https://tinyurl.com/45uwpjem`. Accessed: 2022.

[161] Tuan, T. A., Long, H. V., Kumar, R., Priyadarshini, I., Son, N. T. K., et al. Performance evaluation of botnet ddos attack detection using machine learning. *Evolutionary Intelligence* (2019), 1–12.

[162] Veitch, D., Hohn, N., and Abry, P. Multifractality in tcp/ip traffic: the case against. *Computer Networks 48*, 3 (2005), 293–313.

[163] Wagner, D., Kopp, D., Wichtlhuber, M., Dietzel, C., Hohlfeld, O., Smaragdakis, G., and Feldmann, A. United we stand: Collaborative detection and mitigation of amplification ddos attacks at scale. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security* (2021), pp. 970–987.

[164] Wang, A., Chang, W., Chen, S., and Mohaisen, A. Delving into internet ddos attacks by botnets: characterization and analysis. *IEEE/ACM Transactions on Networking 26*, 6 (2018), 2843–2855.

[165] Wang, A., Mohaisen, A., and Chen, S. An adversary-centric behavior modeling of DDoS attacks. In *Proceedings of the International Conference on Distributed Computing Systems* (2017), IEEE, pp. 1126–1136.

[166] Wang, W., Wu, X. C., Tammana, P., Chen, A., and Ng, T. E. Closed-loop network performance monitoring and diagnosis with SpiderMon. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 267–285.

[167] Wang, Y., Li, D., Lu, Y., Wu, J., Shao, H., and Wang, Y. Elixir: A high-performance and low-cost approach to managing Hardware/Software hybrid flow tables considering flow burstiness. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 535–550.

[168] Welzel, A., Rossow, C., and Bos, H. On measuring the impact of ddos botnets. In *Proceedings of the Seventh European Workshop on System Security* (2014), pp. 1–6.

[169] Wendt, H., and Abry, P. Multifractality tests using bootstrapped wavelet leaders. *IEEE Transactions on Signal Processing 55*, 10 (2007), 4811–4820.

[170] Wendt, H., Abry, P., and Jaffard, S. Bootstrap for empirical multifractal analysis. *IEEE signal processing magazine 24*, 4 (2007), 38–48.

[171] Willinger, W., Taqqu, M. S., Sherman, R., and Wilson, D. V. Self-similarity through high-variability: statistical analysis of ethernet lan traffic at the source level. *IEEE/ACM Transactions on networking 5*, 1 (1997), 71–86.

[172] Xiao, Q., Tang, Z., and Chen, S. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In *Proceedings of IEEE INFOCOM* (2020), IEEE, pp. 974–983.

[173] Xing, J., Hsu, K.-F., Kadosh, M., Lo, A., Piasetzky, Y., Krishnamurthy, A., and Chen, A. Runtime programmable switches. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2022), pp. 651–665.

[174] Xing, J., Wu, W., and Chen, A. Ripple: A programmable, decentralized link-flooding defense against adaptive adversaries. In *Proceedings of the USENIX Security Symposium* (2021).

[175] Xu, Z., Ramanathan, S., Rush, A., Mirkovic, J., and Yu, M. Xatu: boosting existing ddos detection systems using auxiliary signals. In *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies* (2022), pp. 1–17.

[176] Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., and Uhlig, S. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the ACM SIGCOMM Conference* (2018), ACM, pp. 561–575.

[177] Yin, Y., Lin, Z., Jin, M., Fanti, G., and Sekar, V. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM Conference* (2022), pp. 458–472.

[178] Yu, M., Jose, L., and Miao, R. Software defined traffic measurement with OpenSketch. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2013), pp. 29–42.

[179] Yuan, L., Chuah, C.-N., and Mohapatra, P. ProgME: towards programmable network measurement. *IEEE/ACM Transactions on Networking 19*, 1 (2011), 115–128.

[180] Yuan, Y., Lin, D., Mishra, A., Marwaha, S., Alur, R., and Loo, B. T. Quantitative network monitoring with NetQRE. In *Proceedings of the ACM SIGCOMM Conference* (2017), ACM, pp. 99–112.

[181] Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., and Wu, J. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *Proceedings of the Network and Distributed System Security Symposium* (2020).

[182] Zhang, Y., Liu, Z., Wang, R., Yang, T., Li, J., Miao, R., Liu, P., Zhang, R., and Jiang, J. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the ACM SIGCOMM Conference* (2021), pp. 207–222.

[183] Zhao, Y., Yang, K., Liu, Z., Yang, T., Chen, L., Liu, S., Zheng, N., Wang, R., Wu, H., Wang, Y., et al. Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation* (2021), pp. 991–1010.

[184] Zheng, H., Tian, C., Yang, T., Lin, H., Liu, C., Zhang, Z., Dou, W., and Chen, G. Flymon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM Conference* (2022), pp. 486–502.

[185] Zhou, Y., Bi, J., Yang, T., Gao, K., Cao, J., Zhang, D., Wang, Y., and Zhang, C. Hypersight: Towards scalable, high-coverage, and dynamic network monitoring queries. *IEEE Journal on Selected Areas in Communications 38*, 6 (2020), 1147–1160.

[186] Zhou, Y., Sun, C., Liu, H. H., Miao, R., Bai, S., Li, B., Zheng, Z., Zhu, L., Shen, Z., Xi, Y., et al. Flow event telemetry on programmable data plane. In *Proceedings of the ACM SIGCOMM Conference* (2020), pp. 76–89.

[187] Zhou, Y., Zhang, D., Gao, K., Sun, C., Cao, J., Wang, Y., Xu, M., and Wu, J. Newton: Intent-driven network traffic monitoring. In *Proceedings of the ACM Conference on emerging Networking EXperiments and Technologies* (2020), pp. 295–308.

[188] Zhu, H., Zhang, Y., Zhang, L., He, G., and Liu, L. Cbfsketch: A scalable sketch framework for high speed network. In *Proceedings of the Seventh International Conference on Advanced Cloud and Big Data* (2019), IEEE, pp. 357–362.

[189] Zhu, X., Wu, G., Zhang, H., Wang, S., and Ma, B. Dynamic count-min sketch for analytical queries over continuous data streams. In *Proceedings of the International Conference on High Performance Computing* (2018), IEEE, pp. 225–234.